Parallel FFT in Julia
Review of FFT

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

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

or

$$\mathbf{y} = F_n \mathbf{x}$$

where entries of Fourier matrix $F_n$ are given by

$$\{F_n\}_{mk} = \omega_n^{mk}$$
Review of FFT (cont.)

\[ y_m = \sum_{k=0}^{\frac{n}{2} - 1} x_{2k} \omega_n^{mk} + \omega_n^m \sum_{k=0}^{\frac{n}{2} - 1} x_{2k+1} \omega_n^{mk} \]

\[ X_m = Y_m + \omega_n^m Z_m \]

\[ X_{m + \frac{n}{2}} = Y_m - \omega_n^m Z_m \]
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.
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 $\sqrt{n} \times \sqrt{n}$ matrix
- Local computation of sub-problems
- Transpose input
- Local computation of remaining problems
Parallel Transpose Algorithm

\[ s = 4 \]
\[ s = 3 \]
\[ s = 2 \]
\[ s = 1 \]
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

\[ \text{FFT}_\text{Parallel} \left( \text{array} \right) \]

- Bit reverse input array and distribute equally

- for each processor
  - @spawnat proc FFT-Sequential \left( \text{local array} \right) \]

- 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 $\Rightarrow$ 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

<table>
<thead>
<tr>
<th>$2^N$</th>
<th>FFTW</th>
<th>Binary Exchange</th>
<th>Black Box</th>
<th>Communication</th>
<th>Black Box 8p</th>
<th>Communication 8p</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{10}$</td>
<td>0.0002</td>
<td>11.938756</td>
<td>0.10948204</td>
<td>0.150836</td>
<td>0.37363982</td>
<td>0.4737592</td>
</tr>
<tr>
<td>$2^{14}$</td>
<td>0.0009</td>
<td>193.102961</td>
<td>0.18138098</td>
<td>0.1402709</td>
<td>0.59792995</td>
<td>0.416522</td>
</tr>
<tr>
<td>$2^{20}$</td>
<td>0.128</td>
<td>A year?</td>
<td>8.28874588</td>
<td>1.1951251</td>
<td>9.86574506</td>
<td>1.701771</td>
</tr>
<tr>
<td>$2^{25}$</td>
<td>6.3023</td>
<td>Apocalypse</td>
<td>290.634222</td>
<td>30.29218</td>
<td>314.73069</td>
<td>44.75283</td>
</tr>
</tbody>
</table>
Benchmark Results

![Benchmark Results Graph](image)
Communication issues

Issue:

- Redistribution done separately at each stage of the last $\log_2 p$ stages, total $\sum_{k=1}^{\log_2 p} 2^{k-1}$ calls

- Could save overhead by eliminating these calls

Attempted solution:

- Bundle into one redistribution phase $\Rightarrow$ total $\log_2 p$ calls
### New Results

<table>
<thead>
<tr>
<th>$2^{10}$</th>
<th>FFTW</th>
<th>Black Box Improved Redistribution</th>
<th>Communication for Improved Redistribution</th>
<th>Black Box Improved Redistribution 8p</th>
<th>Communication for Improved Redistribution 8p</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.0002</td>
<td>0.10238204</td>
<td>0.08972</td>
<td>0.154734</td>
<td>0.145179</td>
</tr>
<tr>
<td>$2^{14}$</td>
<td>0.0009</td>
<td>0.247728</td>
<td>2.703539</td>
<td>0.284916</td>
<td>0.191254</td>
</tr>
<tr>
<td>$2^{20}$</td>
<td>0.128</td>
<td>8.389301</td>
<td>0.783675</td>
<td>8.717208</td>
<td>1.014697</td>
</tr>
<tr>
<td>$2^{25}$</td>
<td>6.3023</td>
<td>287.0166</td>
<td>21.3901</td>
<td>287.4002</td>
<td>26.08323</td>
</tr>
</tbody>
</table>
New Results

![Graph showing data points and lines with legend for different categories like FFTW, Black Box, Communication, etc.](image-url)
More issues and considerations

- Leverage of sequential FFTW on black box problems
- A separate algorithm, better data distribution?
Questions