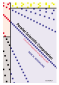
Weights for the FFT
(PSC §3.5)

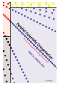


Sequential computation of weights

- The weights of the FFT are the powers of ω_n that are needed in the FFT computation: $1, \omega_n, \omega_n^2, \ldots, \omega_n^{n/2-1}$.
- We can compute these powers by

$$\omega_n^j = e^{-2\pi ij/n} = \cos\frac{2\pi j}{n} - i\sin\frac{2\pi j}{n}.$$

- Computing the weights by successive multiplication $\omega_n^{j+1} = \omega_n \cdot \omega_n^j$ is less accurate and not recommended.
- Typically, computing a sine or cosine costs 10 flops in double precision accuracy. If we compute a weight each time we need it, we perform 20 flops extra for every 10 flops (complex $*, +, -$) in the inner loop of the FFT. This would triple the total cost.
- Alternative: compute weights once, store them in a table.



Using symmetry to compute weights faster

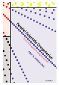
- We can save half the computations by using

$$\omega_n^{n/2-j} = e^{-2\pi i(n/2-j)/n} = e^{-\pi i} e^{2\pi ij/n} = -\overline{(\omega_n^j)}.$$

  Thus, we only need to compute $1, \omega_n, \omega_n^2, \ldots, \omega_n^{n/4}$.
- Taking negatives and complex conjugates is extremely cheap.
- Similarly, we can halve the work again by using

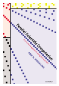
$$\omega_n^{n/4-j} = -i\overline{(\omega_n^j)}.$$

  Now, we only need to compute $1, \omega_n, \omega_n^2, \ldots, \omega_n^{n/8}$.
- The total cost of the weight initialisations is thus about $20 \cdot n/8 = 2.5n$ flops.



Weights for parallel computation

- A brute-force approach: store the complete table of weights on every processor.
- This approach is nonscalable in memory: in the sequential case, we store n vector components and $n/2$ weights. In the parallel case, n/p vector components and $n/2$ weights per processor.
- Furthermore, for small n or large p, the $2.5n$ flops of the weight initialisation may be much more than the $(5n \log_2 n)/p$ local flops of the FFT.
- Some replication of weights is inevitable: stages $k = 2, 4, \ldots, n/p$ are the same on all processors and hence need the same weights.
- Our goal is to find a memory-scalable approach that adds only a few flops to the overall count.



Generalised Discrete Fourier Transform

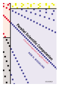
- The Generalised Discrete Fourier Transform (GDFT) is defined by

$$y_k = \sum_{j=0}^{n-1} x_j \omega_n^{j(k+\alpha)}, \text{ for } 0 \leq k < n,$$

  where α is a fixed real parameter.
- GDFT = DFT for $\alpha = 0$.
- We can derive a GFFT, similar to the FFT.
- We can also generalise our matrix notation and obtain a generalised Cooley-Tukey decomposition for the matrix F_n^α defined by

$$(F_n^\alpha)_{jk} = \omega_n^{j(k+\alpha)}.$$

Weights



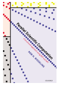
Generalised results—without words

$$\Omega_n^\alpha = \mathrm{diag}(\omega_{2n}^{\alpha}, \omega_{2n}^{1+\alpha}, \omega_{2n}^{2+\alpha}, \ldots, \omega_{2n}^{n-1+\alpha})$$

$$B_n^\alpha = \begin{bmatrix} I_{n/2} & \Omega_{n/2}^\alpha \\ I_{n/2} & -\Omega_{n/2}^\alpha \end{bmatrix}$$

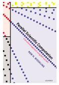
$$F_n^\alpha = B_n^\alpha (I_2 \otimes F_{n/2}^\alpha) S_n$$

$$F_n^\alpha = (I_1 \otimes B_n^\alpha)(I_2 \otimes B_{n/2}^\alpha)(I_4 \otimes B_{n/4}^\alpha) \cdots (I_{n/2} \otimes B_2^\alpha) R_n$$



Aim: reformulating the parallel FFT

We try to express the parallel FFT in sequential GFFTs with suitable α. The α-values may be different on different processors.



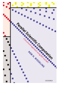
Inner loop in GDFT lingo

```
for $j := j_0$ to $\frac{k}{2} - 1$ step $c$ do
    $\tau := \omega_k^j x_{rk+j+k/2}$;
    $x_{rk+j+k/2} := x_{rk+j} - \tau$;
    $x_{rk+j} := x_{rk+j} + \tau$;
```

This loop takes a local subvector $x(rk + k/2 + j_0 : c : (r+1)k - 1)$ of length $\frac{k}{2c}$, multiplies it by the diagonal matrix

$$\begin{aligned} & \mathrm{diag}(\omega_k^{j_0}, \omega_k^{c+j_0}, \omega_k^{2c+j_0}, \ldots, \omega_k^{k/2-c+j_0}) \\ = \; & \mathrm{diag}(\omega_{k/c}^{j_0/c}, \omega_{k/c}^{1+j_0/c}, \omega_{k/c}^{2+j_0/c}, \ldots, \omega_{k/c}^{k/(2c)-1+j_0/c}) \\ = \; & \Omega_{k/(2c)}^{j_0/c}, \end{aligned}$$

adds it to $x(rk + j_0 : c : rk + k/2 - 1)$, and subtracts it.



Weights

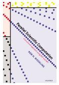
In matrix notation

```
for $r := j_2 \cdot nblocks$ to $(j_2+1) \cdot nblocks - 1$ do
    for $j := j_0$ to $\frac{k}{2} - 1$ step $c$ do
        $\tau := \omega_k^j x_{rk+j+k/2}$;
        $x_{rk+j+k/2} := x_{rk+j} - \tau$;
        $x_{rk+j} := x_{rk+j} + \tau$;
```

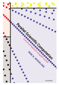
- In the inner loop, the local subvector $x(rk + j_0 : c : (r+1)k - 1)$ is multiplied by $B_{k/c}^{j_0/c}$.
- In the outer loop, the same generalised butterfly is performed for all $nblocks = \frac{nc}{kp}$ local subvectors, thus computing

$$(I_{\frac{nc}{kp}} \otimes B_{k/c}^{j_0/c}) \cdot x(j_2 \frac{nc}{p} + j_0 : c : (j_2+1)\frac{nc}{p} - 1).$$

This is a local computation.



Real butterflies



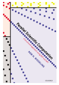
Butterflies form an unordered GFFT

- A complete sequence of butterfly stages is a sequence of maximal length, $k = 2c, 4c, \ldots, \frac{n}{p}c$.
- If we multiply the corresponding matrices $I_{\frac{nc}{kp}} \otimes B_{k/c}^{j_0/c}$ from right to left, we obtain

$$(I_1 \otimes B_{n/p}^{j_0/c})(I_2 \otimes B_{n/(2p)}^{j_0/c}) \cdots (I_{\frac{n}{2p}} \otimes B_2^{j_0/c}) = F_{n/p}^{j_0/c} R_{n/p},$$

  which is an unordered GFFT with parameter
  $\alpha = j_0/c = (s \bmod c)/c$.
- Note the dependence on the processor number s.

Weights



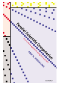
An incomplete sequence is OK at the start

- For $c = 1$, we have $j_0 = s \bmod c = 0$, so that all factors have the form $I_{\frac{nc}{kp}} \otimes B_{k/c}^{j_0/c} = I_{\frac{n}{kp}} \otimes B_k$.
- Now we do not need a complete sequence to obtain a simple formula: if we multiply the matrices for $k = 2, 4, \ldots, k_1$ from right to left we get

$$\begin{aligned} & (I_{\frac{n}{k_1 p}} \otimes B_{k_1}) \cdots (I_{\frac{n}{4p}} \otimes B_4)(I_{\frac{n}{2p}} \otimes B_2) \\ = \quad & I_{\frac{n}{k_1 p}} \otimes ((I_1 \otimes B_{k_1}) \cdots (I_{\frac{k_1}{4}} \otimes B_4)(I_{\frac{k_1}{2}} \otimes B_2)) \\ = \quad & I_{\frac{n}{k_1 p}} \otimes (F_{k_1} R_{k_1}). \end{aligned}$$

- We restructure our algorithm, modifying the c-loop so that we start with one incomplete sequence, and then execute the remainder with complete sequences.

Weights



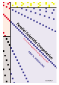
Number of iterations at the start

- We have $t + 1$ iterations, where

$$c = 1, k_1, k_1\frac{n}{p}, \ldots, k_1\left(\frac{n}{p}\right)^{t-1} = p.$$

- Thus, k_1 is given by

$$k_1 = \frac{n}{(n/p)^t}.$$

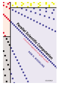


Restructured parallel FFT

```
      { $\mathrm{distr}(\mathbf{x})$ = cyclic }
(0)   bitrev($x(s: p: n-1), n/p$);
      { $\mathrm{distr}(\mathbf{x})$ = block with bit-reversed processor number }

      $t := \lceil \frac{\log_2 p}{\log_2(n/p)} \rceil$; $k_1 := \frac{n}{(n/p)^t}$; $rev := true$;
      for $r := s \cdot \frac{n}{k_1 p}$ to $(s+1) \cdot \frac{n}{k_1 p} - 1$ do
          UFFT($x(rk_1 : (r+1)k_1 - 1), k_1$);
```

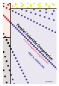
Weights



Restructured parallel FFT

```
        { distr(x) = cyclic }
(0)     bitrev($x(s : p : n-1), n/p$);
        { distr(x) = block with bit-reversed processor number }

        $t := \lceil \frac{\log_2 p}{\log_2 (n/p)} \rceil$; $k_1 := \frac{n}{(n/p)^t}$; $rev := true$;
        for $r := s \cdot \frac{n}{k_1 p}$ to $(s+1) \cdot \frac{n}{k_1 p} - 1$ do
            UFFT($x(rk_1 : (r+1)k_1 - 1), k_1$);
        $c_0 := 1$; $c := k_1$;
        while $c \leq p$ do
(1)         redistr($\mathbf{x}, n, p, c_0, c, rev$);
            { distr(x) = group-cyclic with cycle $c$ }
(2)         $j_0 := s \bmod c$; $j_2 := s \operatorname{div} c$; $rev := false$;
            UGFFT($x(j_2 \frac{nc}{p} + j_0 : c : (j_2+1)\frac{nc}{p} - 1), n/p, j_0/c$);
            $c_0 := c$; $c := \frac{n}{p} c$;
        { distr(x) = cyclic }
```



Weights

A different way of computing the GDFT

- We can rewrite the ordered GDFT as

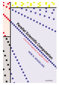
$$y_k = \sum_{j=0}^{n-1} (x_j \omega_n^{j\alpha}) \omega_n^{jk}.$$

- Thus, we can multiply the components of the input vector first by scalar factors and then perform a DFT.
- In matrix language, define the twiddle matrix

$$T_n^\alpha = \mathrm{diag}(1, \omega_n^\alpha, \omega_n^{2\alpha}, \ldots, \omega_n^{(n-1)\alpha}),$$

giving $F_n^\alpha = F_n T_n^\alpha$.
- For an unordered GDFT, we twiddle with $R_n T_n^\alpha R_n$.
- Twiddling costs n/p extra complex multiplications, or $6n/p$ flops, in every computation superstep except the first.

Weights

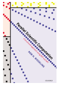


Memory needed by the parallel FFT

- The total amount of memory space per processor in reals used by the parallel FFT is

$$M_{\mathrm{FFT}} = \left(2 \cdot \left\lceil \frac{\log_2 p}{\log_2(n/p)} \right\rceil + 3\right) \cdot \frac{n}{p}.$$

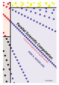
- This is for:
  - n/p complex vector components;
  - $n/(2p)$ complex weights of an FFT of length n/p;
  - n/p complex twiddle factors for each of the $\lceil \frac{\log_2 p}{\log_2(n/p)} \rceil$ GFFTs performed locally.



Memory scalability

- We call the memory requirements of a BSP algorithm scalable if
$$M(n,p) = \mathcal{O}\left(\frac{M_{\text{seq}}(n)}{p} + p\right).$$
- Motivation of the $\mathcal{O}(p)$ term: BSP algorithms are based on all-to-all communication supersteps, where each processor deals with $p-1$ others, and needs already $\mathcal{O}(p)$ buffer memory for storing communication meta-data.

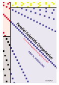
Weights



The parallel FFT is memory-scalable

- For $p \le n/p$, only one twiddle array has to be stored, so that the total memory requirement is $M(n,p) = 5n/p$, which is of the right order.
- For $p > n/p$, we need $t-1$ additional iterations, each requiring a twiddle array. Fortunately, the total extra twiddle memory is at most

$$\begin{aligned} \frac{2(t-1)n}{p} &= 2\left(\frac{n}{p} + \frac{n}{p} + \cdots + \frac{n}{p}\right) \\ &\le 2\frac{n}{p} \cdot \frac{n}{p} \cdots \frac{n}{p} \\ &= 2\left(\frac{n}{p}\right)^{t-1} = \frac{2p}{k_1} \le p. \end{aligned}$$



Weights

Summary

- We have introduced the Generalised Discrete Fourier Transform defined by

$$y_k = \sum_{j=0}^{n-1} x_j \omega_n^{j(k+\alpha)}.$$

- We have restructured our parallel algorithm, expressing the local computations as sequential GFFTs.
- The sequential GFFTs can be performed at little extra cost by multiplying the local vector first by a diagonal twiddle matrix, and then performing an unordered FFT.
- The restructured algorithm is memory-scalable, with

$$M(n,p) = \mathcal{O}\left(\frac{M_{\text{seq}}(n)}{p} + p\right).$$

Weights