# Parallel Numerical Algorithms

Chapter 6 – Matrix Models
Section 6.1 – Fast Fourier Transform

Michael T. Heath and Edgar Solomonik

Department of Computer Science
University of Illinois at Urbana-Champaign

CS 554 / CSE 512

## Outline

## Convolution

Convolution takes input $\boldsymbol{a}$ and $\boldsymbol{b}$ and computes $\boldsymbol{c}$

$$\forall k \in [0, n-1] \quad c_k = \sum_{j=0}^{k} a_j b_{k-j}$$

- If $\boldsymbol{a}$ and $\boldsymbol{b}$ are coefficients of degree $n/2-1$ polynomials

$$p_a(x) = \sum_{k=0}^{n/2-1} a_k x^k, \quad p_b(x) = \sum_{k=0}^{n/2-1} b_k x^k$$

  the convolution computes the coefficients $\boldsymbol{c}$ of the product

$$p_c(x) = p_a(x) p_b(x) = \sum_{k=0}^{n-1} c_k x^k$$

- naive evaluation costs $O(n^2)$ operations

## Convolution and Toeplitz Matrices

- Convolution can be interpreted as matrix-vector multiplication with a triangular *Toeplitz matrix*

$$[c_0\ c_1\ c_2\ c_3] = [a_1\ a_2\ a_3\ a_4] \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ 0 & b_0 & b_1 & b_2 \\ 0 & 0 & b_0 & b_1 \\ 0 & 0 & 0 & b_0 \end{bmatrix}$$

- *Toeplitz* and *Hankel* matrices (in the latter, each antidiagonal is defined by a single element) provide a general matrix representation for convolutional operators

## Convolution via Interpolation (Toom-Cook)

- Evaluate $p_a$ and $p_b$ at a set of nodes $x_0, \dots, x_{n-1}$
- The values of $p_c$ at each $x_j$ are then easily obtained

$$p_c(x_j) = p_a(x_j) p_b(x_j)$$

- The inverse DFT, $\boldsymbol{F}_n^{-1} p_c(\boldsymbol{x})$ interpolates the values of the polynomial $p_c$ at each $x_j$ producing its coefficients $\boldsymbol{c}$
- For a Vandermonde matrix $\boldsymbol{V}_n$ associtated with nodes $x_0, \dots, x_{n-1}$, the overall procedure is described by

$$\boldsymbol{c} = \boldsymbol{V}_n^{-1}[(\boldsymbol{V}_n \boldsymbol{a}) \odot (\boldsymbol{V}_n \boldsymbol{b})]$$

where $\odot$ is an elementwise product ($\boldsymbol{a}$ and $\boldsymbol{b}$ are padded with trailing zeros)

## Roots of Unity

- For given integer $n$, we use notation

$$\omega_n = \cos(2\pi/n) - i\sin(2\pi/n) = e^{-2\pi i/n}$$

  for primitive $n$th root of unity, where $i = \sqrt{-1}$

- $n$th roots of unity, sometimes called *twiddle factors* in this context, are then given by $\omega_n^k$ or by $\omega_n^{-k}$, $k = 0, \ldots, n-1$

- For convenience, we will assume that $n$ is power of two, and all logarithms used will be base two

- We will also index sequences (components of vectors) starting from $0$ rather than $1$

## Discrete Fourier Transform

- *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 *DFT matrix* $\boldsymbol{F}_n$ are given by

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

- The DFT matrix is a Vandermonde matrix with nodes $\omega_n^0, \ldots, \omega_{n-1}^n$

## Inverse DFT

- It is easily seen that

$$\boldsymbol{F}_n^{-1} = (1/n)\boldsymbol{F}_n^H$$

- Hence $\kappa(\boldsymbol{F}_n) = 1$, while for Vandermonde matrices with real nodes, condition number grows exponentially with $n$

- So, since $(\omega_n^k)^* = \omega_n^{-k}$, *inverse DFT* is given by

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

## Example

$$\boldsymbol{F}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}$$

$$4\,\boldsymbol{F}_4^{-1} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}$$

## Radix-2 Fast Fourier Transform (FFT)

- Consider $\boldsymbol{b} = \boldsymbol{F}_n \boldsymbol{a}$, we have

$$\forall j \in [0, n-1] \quad b_j = \sum_{k=0}^{n-1} \omega_n^{jk} a_k$$

- Express DFT as two DFTs of dimension $n/2$, with a different root of unity $\omega_{n/2}$
- Separate summands into odds and evens, use $\omega_{n/2} = \omega_n^2$

$$\begin{aligned} b_j &= \sum_{k=0}^{n/2-1} \omega_n^{j(2k)} a_{2k} + \sum_{k=0}^{n/2-1} \omega_n^{j(2k+1)} a_{2k+1} \\ &= \sum_{k=0}^{n/2-1} \omega_{n/2}^{jk} a_{2k} + \omega_n^j \sum_{k=0}^{n/2-1} \omega_{n/2}^{jk} a_{2k+1} \end{aligned}$$

## Radix-2 Fast Fourier Transform (FFT), contd.

$$b_j = \underbrace{\sum_{k=0}^{n/2-1} \omega_{n/2}^{jk} a_{2k}}_{u_j} + \omega_n^j \underbrace{\sum_{k=0}^{n/2-1} \omega_{n/2}^{jk} a_{2k+1}}_{v_j}$$

The summations for $b_j$ and $b_{j+n/2}$ are closely related,

$$b_{j+n/2} = \sum_{k=0}^{n/2-1} \omega_{n/2}^{(j+n/2)k} a_{2k} + \omega_n^{j+n/2} \sum_{k=0}^{n/2-1} \omega_{n/2}^{(j+n/2)k} a_{2k+1}$$

Now $\omega_{n/2}^{(j+n/2)k} = \omega_{n/2}^{jk}$ since $(\omega_{n/2}^{n/2})^k = 1^k = 1$ and using $\omega_n^{n/2} = -1$,

$$b_{j+n/2} = \underbrace{\sum_{k=0}^{n/2-1} \omega_{n/2}^{jk} a_{2k}}_{u_j} - \omega_n^j \underbrace{\sum_{k=0}^{n/2-1} \omega_{n/2}^{jk} a_{2k+1}}_{v_j}$$

## Radix-2 Fast Fourier Transform (FFT), contd.

- Let vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ be two recursive FFTs, $\forall j \in [0, n/2-1]$

$$u_j = \sum_{k=0}^{n/2-1} \omega_{n/2}^{jk} a_{2k}, \quad v_j = \sum_{k=0}^{n/2-1} \omega_{n/2}^{jk} a_{2k+1}$$

- Given $\boldsymbol{u}$ and $\boldsymbol{v}$ scale using "twiddle factors" $z_j = \omega_n^j \cdot v_j$
- Then it suffices to combine the vectors as follows $\boldsymbol{b} = \begin{bmatrix} \boldsymbol{u}+\boldsymbol{z} \\ \boldsymbol{u}-\boldsymbol{z} \end{bmatrix}$
- This recombination is an FFT of dimension 2

$$\boldsymbol{b} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{bmatrix} = \operatorname{vec}\left(\begin{bmatrix} \boldsymbol{b}_1 & \boldsymbol{b}_2 \end{bmatrix}\right) = \operatorname{vec}\left(\begin{bmatrix} \boldsymbol{u} & \boldsymbol{z} \end{bmatrix} \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}}_{\boldsymbol{F_4}[0:2,0:2]}\right)$$

- Radix-$r$ algorithm for any $\boldsymbol{A} \in \mathbb{R}^{r \times n/r}$

$$\boldsymbol{F}_n \operatorname{vec}(\boldsymbol{A}) = \operatorname{vec}\left(\left([\boldsymbol{F}_n[0:r, 0:n/r] \odot (\boldsymbol{F}_r \boldsymbol{A})]\boldsymbol{F}_{n/r}\right)^T\right)$$

## FFT Algorithm

```
procedure fft($x, y, n, \omega$)
if $n = 1$ then
    $y[0] = x[0]$
else
    for $k = 0$ to $(n/2) - 1$
        $p[k] = x[2k]$
        $s[k] = x[2k + 1]$
    end
    fft($p, q, n/2, \omega^2$)
    fft($s, t, n/2, \omega^2$)
    for $k = 0$ to $n - 1$
        $y[k] = q[k \bmod (n/2)] + \omega^k t[k \bmod (n/2)]$
    end
end
```

## Complexity of FFT Algorithm

- In the radix-2 algorithm, there are $\log n$ levels of recursion (depth), each of which involves $\Theta(n)$ arithmetic operations, so total cost is $\Theta(n\ \log n)$
- By contrast, straightforward evaluation of matrix-vector product defining DFT requires $\Theta(n^2)$ arithmetic operations.
- Alternatively, setting the radix to be the square root of the vector dimension at each step, yields $2\sqrt{n}$ FFTs of size $\sqrt{n}$, $O(\log\log n)$ levels of recursion with overall work,

$$Q(n) = 2\sqrt{n}Q(\sqrt{n}) + O(n) = O(n\log n),$$

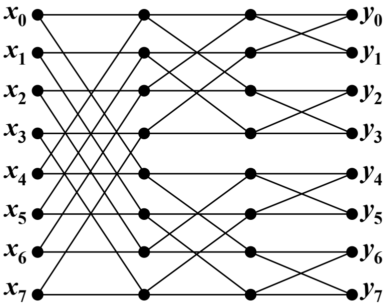
  and due to the dependency among left and right FFTs, depth $D(n) = 2D(\sqrt{n}) + 1 = O(\log n)$.

## Computing Inverse DFT

- Because of similar form of DFT and its inverse, FFT algorithm can also be used to compute inverse DFT efficiently
- Ability to transform back and forth quickly between time and frequency domains makes it practical to perform any computations or analysis that may be required in whichever domain is more convenient and efficient

## Binary Exchange Parallel FFT

- To obtain fine-grain decomposition of FFT, we assign input data $x_k$ to task $k$, which also computes result $y_k$



- At stage $m$ of algorithm, tasks $k$ and $j$ exchange data, where $k$ and $j$ differ only in their $m$th bits

## Binary Exchange Parallel FFT

- There are $n$ tasks and $\log n$ stages, so parallel time required to compute FFT is

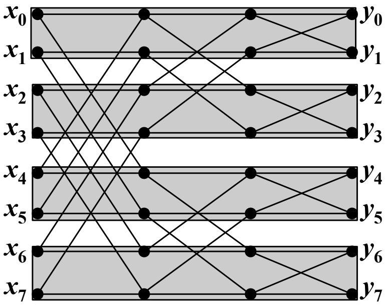
$$T_n = (\gamma + \alpha + \beta)\,\log n$$

  where $\gamma$ is cost of multiply-add, and $\alpha + \beta$ is cost of exchanging one number between pair of tasks at each stage

- Hypercube is natural network for FFT algorithm

## Binary Exchange Parallel FFT

- 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 Parallel FFT

- Components having their $\log p$ most significant bits in common are assigned to same task
- 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 Parallel FFT

- Each stage involves updating of $n/p$ components by each task, and exchange of $n/p$ components for each of first $\log p$ stages
- Thus, total time required using hypercube network is

$$T_p = \alpha\,(\log p) + \beta\,n\,(\log p)/p + \gamma\,n\,(\log n)/p$$

- To determine isoefficiency function, set

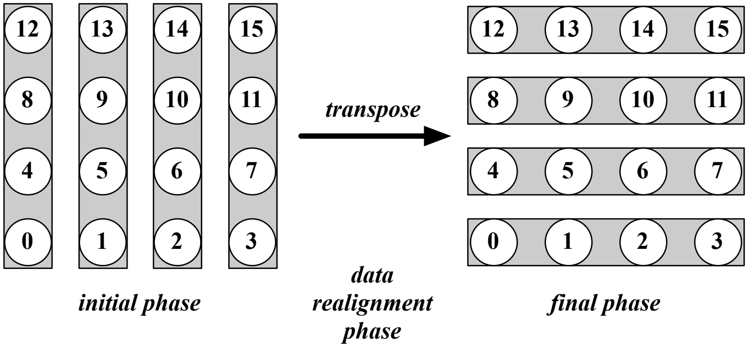
$$\gamma\,n\,\log n \approx E\,(\alpha\,p\,\log p + \beta\,n\,\log p + \gamma\,n\,\log n)$$

which holds if $n = \Theta(p)$, so isoefficiency function is $\Theta(p\,\log p)$, since $T_1 = \Theta(n\,\log n)$

## Transpose Parallel FFT

- Binary exchange algorithm has one phase that is communication free and another phase that requires communication at each stage
- Another approach is to realign data so that both computational phases are communication free, and only communication is for data realignment phase between computational phases
- To accomplish this, data can be organized in $\sqrt{n} \times \sqrt{n}$ array, as illustrated next for $n = 16$

## Transpose Parallel FFT



## Transpose Parallel FFT

- If array is partitioned by columns, which are assigned to $p \leq \sqrt{n}$ tasks, then no communication is required for first $\log(\sqrt{n}\,)$ stages
- Data are then transposed using all-to-all personalized collective communication, so that each *row* of data array is now stored in single task
- Thus, final $\log(\sqrt{n}\,)$ stages now require no communication
- Overall performance of transpose algorithm depends on particular implementation of all-to-all personalized collective communication

## Transpose Parallel FFT

- Taking into account the cost of an all-to-all, $O(\alpha \log p + \beta n \log(p)/p)$, transpose-based approach yields total parallel time

$$T_p = \Theta(\alpha\, \log p + \beta\, n \log(p)/p + \gamma\, n\, \log(n)/p)$$

- This time is the same as for the binary exchange parallel FFT, the only advantage being that all communication happens at once (can be performed with a single collective communication routine).

## References

- A. Averbuch and E. Gabber, Portable parallel FFT for MIMD multiprocessors, *Concurrency: Practice and Experience* 10:583-605, 1998
- C. Calvin, Implementation of parallel FFT algorithms on distributed memory machines with a minimum overhead of communication, *Parallel Computing* 22:1255-1279, 1996
- R. M. Chamberlain, Gray codes, fast Fourier transforms, and hypercubes, *Parallel Computing* 6:225-233, 1988
- E. Chu and A. George, FFT algorithms and their adaptation to parallel processing, *Linear Algebra Appl.* 284:95-124, 1998

## References

- A. Edelman, Optimal matrix transposition and bit reversal on hypercubes: all-to-all personalized communication, *J. Parallel Computing* 11:328-331, 1991
- A. Gupta and V. Kumar, The Scalability of FFT on Parallel Computers, *IEEE Trans. Parallel Distrib. Sys.* 4:922-932, 1993
- R. B. Pelz, Parallel FFTs, D. E. Keyes, A. Sameh, and V. Venkatakrishnan, eds., *Parallel Numerical Algorithms*, pp. 245-266, Kluwer, 1997
- P. N. Swarztrauber, Multiprocessor FFTs, *Parallel Computing* 5:197-210, 1987
- J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM Philadelphia, 1997.