CS 450: Numerical Anlaysis

Lecture 29 Chapter 12 Fast Fourier Transform

Fast Solvers: Multigrid and FFT

Edgar Solomonik

Department of Computer Science
University of Illinois at Urbana-Champaign

May 2, 2018

Sparse Linear Systems and Time-independent PDEs

- The Poisson equation serves as a model problem for numerical methods:

$$\underbrace{A}_{I \otimes T + T \otimes I} x = b \qquad A \in \mathbb{R}^{n \times n}$$

where $T = \begin{bmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & & -1 \\ & & -1 & 2 \end{bmatrix}$

- Dense, sparse direct, iterative, FFT, and Multigrid methods provide increasingly good complexity for the problem:

naive dense $O(n^3)$

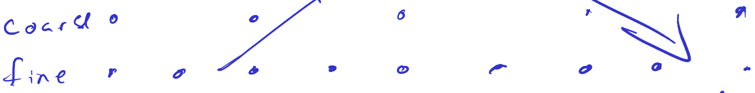
sparse direct (Cholesky w/ nested dissection) $O(n^{3/2})$ cost, $O(n\log(n))$ memory usage

Conjugate Gradient $O(n^{3/2})$ cost $O(n)$ memory

FFT $O(n\log n)$, Multigrid $O(n)$

Multigrid

- Multigrid employs a hierarchy of grids to accelerate iterative methods:



1st resolve error on fine grid by 'smoothing'

e.g. application of CG / Jacobi

restrict the residual equation $\underbrace{A\hat{x} = r}_{\text{solve using approx. on coarse}} = A\underbrace{x}_{\text{current approximation on the fine grid}} - b$

$x + \hat{x}$

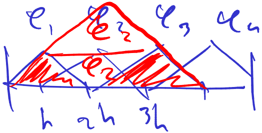
- The multigrid method works by resolving high-frequency error components on finer-grids and low-frequency error components on coarser grids:

finer-grids → correct high-frequency

coarse-grid → correct low-frequency comp. error

Multigrid

- Consider the Galerkin approximation with linear finite elements to the Poisson equation $u'' = f(t)$ with boundary conditions $u(a) = u(b) = 0$:



$$\phi_i^{(h)}(t) = \begin{cases} (t - t_{i-1})/h & : t \in [t_{i-1}, t_i] \\ (t_{i+1} - t)/h & : t \in [t_i, t_{i+1}] \\ 0 & : \text{otherwise} \end{cases}$$

where $t_0 = t_1 = a$ and $t_{n+1} = t_n = b$.

$$\int f(t)\, \varphi_i^{(h)}(t)\, dt = -\sum_j x_j \int \underbrace{\varphi_i^{(h)'}(t)\, \varphi_j^{(h)'}(t)}_{a_{ij}^{(h)}}\, dt$$

$a_{ij}^{(h)}$ – stiffness matrix

$$\varphi_i^{(2h)}(t) = \frac{1}{2}\left(\varphi_{2i-2}^{(h)}(t) + 2\varphi_{2i-1}^{(h)}(t) + \varphi_{2i}^{(h)}(t)\right)$$

= standard $2h$ basis

Coarse Grid Matrix

- Multigrid restricts the residual equation on the fine grid $\boldsymbol{A}^{(h)}\boldsymbol{x} = \boldsymbol{r}^{(h)}$ to the coarse grid:

$$e^{(h)} = \begin{bmatrix} e_1^{(h)} \\ e_2^{(h)} \\ e_3^{(h)} \\ \vdots \end{bmatrix}$$

basis

$$e^{(2h)} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 & & \\ & 1 & 2 & 1 & \\ & & & \ddots & \end{bmatrix} e^{(h)}$$

P – restriction matrix in multigrid

$$x^{(h)} = P\hat{x}^{(2h)}$$

$$x^T e^{(2h)} = \underbrace{x^T P}_{\text{coefficients in fine grid}} e^{(h)}$$

coefficients in coarse

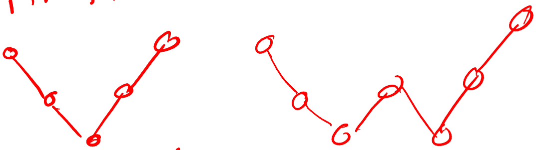
$$A^{(h)}\hat{x}^{(h)} = r^{(h)}$$

$$PA^{(h)}P^T\hat{x}^{(2h)} = Pr^{(h)}$$

$$\big(\underbrace{PA^{(h)}P^T}_{A^{(2h)}}\big)_{ij} = \int e_i^{(2h)\prime}(t)\, e_j^{(2h)\prime}(t)\, dt = P_i \int e^{(h)\prime\,T} \ldots P_j^T e \ldots$$

1. smoothing
2. restriction
3. recursive solve on coarse grid
4. interpolation of coarse grid solution
5. smoothing

finest



coarsest

$\log(n)$ levels, $O\left(\frac{n}{2^i}\right)$ work on the ith level, overall $O(n)$

Restricting the Residual Equation

- Given the fine-grid residual $\boldsymbol{r}^{(h)}$, we seek to use the coarse grid to approximate $\boldsymbol{x}^{(h)}$ so that $\boldsymbol{A}\boldsymbol{x}^{(h)} \approx \boldsymbol{r}^{(h)}$

Discrete Fourier Transform

Vandermonde matrix V, with $v_{ij} = (\omega_{(n)}^i)^j = \omega_{(n)}^{ij}$

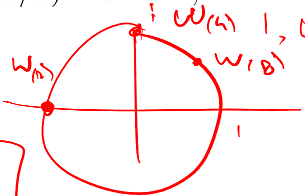
- The solutions to hyperbolic PDEs like Poisson are wave-like and take on simple representations in the frequency basis, both for continuous and discretized equations. We define the *discrete Fourier transform* using

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

nodes are $1, \omega_{(n)}, \omega_{(n)}^2, \ldots, \omega_{(n)}^{n-1}$

$F \in \mathbb{R}^{n\times n}$

$\omega_{(n)}^n = 1$



$$f_{ij} = \omega_{(n)}^{ij} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix}$$

$\forall i,j \in [0, n-1]$

symmetric

~~Hermitian~~

$nD^H = D^{-1} \Rightarrow D^*D = nI,$

DFT, $D \cdot v$

Fast Fourier Transform (FFT)

$$w_{(n)}^2 = w_{(n/2)}$$

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

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

the FFT computes this recursively via 2 FFTs of dimension $n/2$, using $\omega_{(n/2)} = \omega_{(n)}^2$,

$$b_j = \sum_{k=1}^{n/2-1} w_{(n)}^{j2k} a_{2k} + \sum_{k=1}^{n/2-1} w_{(n)}^{j(2k+1)} a_{2k+1}$$

$$b_j = \sum_{k=1}^{n/2-1} w_{(n/2)}^{jk} a_{2k} + w_{(n)}^{j} \sum_{k=1}^{n/2-1} w_{(n/2)}^{jk} a_{2k+1}$$

Fast Fourier Transform Derivation

- The FFT leverages similarity between the first and second half of the output,

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

corresponds closely to the entry shifted by $n/2$,

difference

$$b_{j+n/2} = \overbrace{\sum_{k=0}^{n/2-1} \omega_{(n/2)}^{(j+\cancel{n/2})k} a_{2k}}^{u_j} + \omega_{(n)}^{j+n/2} \underbrace{\sum_{k=0}^{n/2-1} \omega_{(n/2)}^{(j+\cancel{n/2})k} a_{2k+1}}_{v_j}$$

$$\omega_{(n/2)}^{(j+n/2)k} = \omega_{(n/2)}^{jk}$$

$$\omega_{(n)}^{j+n/2} = -\omega_{(n)}^{j}$$

FFT Algorithm Summary

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

$$\underbrace{u_j = \sum_{k=0}^{n/2-1} \omega_{(n/2)}^{jk} a_{2k}}_{\text{FFT 1}}, \quad \underbrace{v_j = \sum_{k=0}^{n/2-1} \omega_{(n/2)}^{jk} a_{2k+1}}_{\text{FFT 2}}$$

$$z_j = \omega_{(n)}^{j} v_j$$

$$b = \begin{bmatrix} u + z \\ v - z \end{bmatrix}$$

- The FFT has $O(n \log n)$ cost complexity:

$$T(n) = 2T(n/2) + O(n)$$
$$= O(n \log(n))$$

Applications of the FFT

- We can rapidly multiply degree $n-1$ polynomials by considering their values $\omega_{(n)}^i$ for $i \in \{0, \ldots, n-1\}$

- More generally the DFT can be used to solve any Toeplitz linear system (convolution):

Convolution via DFT

- The Fourier transform method for computing a convolution is given by

$$c_k = \frac{1}{n} \sum_s \omega_{(n)}^{-ks} \Big(\sum_j \omega_{(n)}^{sj} a_j \Big) \Big(\sum_t \omega_{(n)}^{st} b_t \Big)$$

Solving Numerical PDEs with the FFT

- 1D finite-difference schemes on a regular grid correspond to convolutions:

- For the 1D Poisson model problem, the eigenvectors of $\boldsymbol{T}$ corresponds to the imaginary part of a minor of a $2(n+1)$-dimensional DFT matrix:

- Multidimensional Poisson can be handled with multidimensional FFT: