

CMSC878R/AMSC698R: Fast Multipole Methods: Scribe

Notes for Lectures 1 & 2

Instructor for this Lecture: Ramani Duraiswami

Scribe: Hazem El-Alfy

September 9, 2003

1 Basic refresh

1.1 Vectors and Matrices

Recall that an N -dimensional column vector \mathbf{x} and its transpose \mathbf{x}^t are written as:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} \text{ and } \mathbf{x}^t = (x_1 \quad x_2 \quad \cdots \quad x_N)$$

An $M \times N$ matrix \mathbf{A} is written as

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \cdots & a_{MN} \end{pmatrix}$$

The matrix vector product $\mathbf{v} = \mathbf{A}\mathbf{x}$ can be expanded as:

$$\begin{aligned} v_1 &= a_{11}x_1 + a_{12}x_2 + \cdots + a_{1N}x_N \\ v_2 &= a_{21}x_1 + a_{22}x_2 + \cdots + a_{2N}x_N \\ &\dots \\ v_M &= a_{M1}x_1 + a_{M2}x_2 + \cdots + a_{MN}x_N \end{aligned}$$

This product can be viewed as M sums of the form: $v_j = \sum_{i=1}^N a_{ji}x_i$, $j = 1, \dots, M$.

There are N products and sums per line.

A total of M lines.

Thus, the total operation count is $M \times N$ additions and $M \times N$ multiplications to compute N entries.

2 Asymptotic Complexity

The notion of *asymptotic complexity* arises often in the analysis of algorithms as a means of comparing the complexities of different algorithms. Let $f(n)$ and $g(n)$ be two functions. Their asymptotic behavior is compared as $n \rightarrow \infty$.

· Asymptotic equivalence: $f(n) \sim g(n)$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$.

Example: $f(n) = n^2 + 3n$, $g(n) = n^2 + 5$.

· Asymptotically smaller (Little ‘Oh’): $f(n) = o(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$.

Example: $f(n) = 3n$, $g(n) = n^2 + n$.

· Asymptotic order of growth (Big ‘Oh’): $f(n) = O(g(n))$ if $\limsup_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$.

Equivalently, $f(n) = O(g(n))$ if $\exists c, n_0 \geq 0 : \forall n \geq n_0, |f(n)| \leq c \cdot g(n)$.

Example: $f(n) = 5n^2 + 3n$, $g(n) = n^2 + 5$.

If $f = o(g)$ ($\lim = 0$) or $f \sim g$ ($\lim = 1$) then $f = O(g)$ ($\lim < \infty$)

If $f = o(g)$ ($\lim \frac{f}{g} = 0$), then $g \neq O(f)$ ($\lim \frac{g}{f} \rightarrow \infty$).

· Same order of growth (Theta): $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $g(n) = O(f(n))$.

Example: $f(n) = n^2 + 3n$, $g(n) = n^2 + 5$.

· Common complexities:

- - $O(1)$ - Constant amount of time, not proportional to any variable.
- - $O(N)$ - Proportional to the size of N . (e.g. a loop to N , $0.5N$, $25N$, etc...)
- - $O(N^2)$ - Proportional to N^2 . (e.g. two nested loops both proportional to N).
- - $O(\log N)$ - e.g. data of size N is split in 2 halves at each step.
- - $O(N \log N)$ - e.g. $\log N$ splits each requiring work proportional to N .

3 Key Ideas of FMM

The Fast Multipole Method (FMM) is an algorithm for achieving fast products of particular dense matrices with vectors. As mentioned in section 1 above, matrix vector products are a set of sums. So, the FMM can be viewed as an algorithm for fast summations.

The next example introduces some key ideas of the FMM without really being the exact algorithm. Consider the sums:

$$v(y_j) = \sum_{i=1}^N u_i (y_j - x_i)^2, \quad j = 1, \dots, M$$

These sums can be put in matrix form as:

$$\begin{pmatrix} v_1 \\ \vdots \\ v_M \end{pmatrix} = \begin{pmatrix} (y_1 - x_1)^2 & \cdots & (y_1 - x_N)^2 \\ \vdots & \ddots & \vdots \\ (y_M - x_1)^2 & \cdots & (y_M - x_N)^2 \end{pmatrix} \begin{pmatrix} u_1 \\ \vdots \\ u_N \end{pmatrix}$$

The naïve straightforward way to evaluate the matrix vector product (or the sums) obviously requires $O(MN)$ operations. Instead, the sums can be written as:

$$\begin{aligned} v(y_j) &= \left(\sum_{i=1}^N u_i \right) y_j^2 + \left(\sum_{i=1}^N u_i x_i^2 \right) - 2y_j \left(\sum_{i=1}^N u_i x_i \right), \quad j = 1, \dots, M \\ &= \alpha y_j^2 + \beta - 2\delta y_j, \quad j = 1, \dots, M \end{aligned}$$

Each coefficient α, β and δ requires $O(N)$ operations. A total of $3N$ operations. The sums $v(y_j)$ require each 3 operations. A total of $3M$ operations. This sums up to $O(3M + 3N) = O(M + N)$ operations only.

The key idea or "trick" is to change the order of summation provided that the matrix entries are derived from particular functions that can be *factorized* or have their variables separated. This idea is more elaborated in the next example.

Consider the sums, or matrix vector product:

$$v_j = \sum_{i=1}^N u_i \phi(y_j - x_i), \quad j = 1, \dots, M$$

Direct evaluation requires $O(MN)$ operations and $O(N^2)$ space to store the matrix entries. Now, assume that the function ϕ can be represented by a single uniform expansion everywhere (such expansion is not usually available), as follows

$$\phi(y) = \sum_{l=1}^p a_l(x_*) R_l(y - x_*) + \epsilon(p)$$

then, the sum v_j can be written as

$$\begin{aligned} v_j &\cong \sum_{i=1}^N u_i \sum_{l=1}^p a_l g_l(x_i) h_l(y_j), \quad j = 1, \dots, M \\ &= \sum_{l=1}^p h_l(y_j) \sum_{i=1}^N u_i a_l g_l(x_i), \quad j = 1, \dots, M \\ &= \sum_{l=1}^p B_l h_l(y_j), \quad j = 1, \dots, M \end{aligned}$$

For each $l = 1, \dots, p$, compute $B_l = \sum_{i=1}^N u_i a_l g_l(x_i) \rightarrow O(Np)$ operations, and $O(p)$ space.

For each $j = 1, \dots, M$, compute $v_j = \sum_{l=1}^p B_l h_l(y_j) \rightarrow O(Mp)$ operations.

This totals to $O((M + N)p)$ operations, and $O(M + N)$ space to store the points x_i and y_j plus $O(p)$ space to store the coefficients B_l . Usually, M is the same order of magnitude as N and $p \ll N$, so, the "fast method" requires:

$$O(M + N) \text{ operations, and } O(N) \text{ space}$$

4 Simple Applications

4.1 Iterative Methods for Linear Systems

It is required to solve the linear system of equations

$$\mathbf{Ax} = \mathbf{b}$$

where \mathbf{A} is an $N \times N$ matrix, \mathbf{x}, \mathbf{b} are N vectors.

Direct solution of the system, such as Gauss elimination, LU decomposition (Cholesky decomposition) or SVD require $O(N^3)$ operations and $O(N^2)$ space to store the matrix \mathbf{A} . As seen previously, the FMM accelerates matrix vector multiplication. Iterative methods, such as Jacobi iteration, conjugate gradient,

Krylov methods and others, usually converge in k steps, each requiring a few matrix vector products in each step. With $k \ll N$, these methods require $O(kN^2) = O(N^2)$ operations. A fast matrix vector multiplication algorithm will require only $O(kN \log N) = O(N \log N)$ operations and $O(N)$ space.

Integral Equations

Consider the *linear Fredholm* integral equation of the second kind

$$\int k(x, y)u(x)dx + au(y) = f(y)$$

Solving the equation is to find the unknown function $u(y)$. This is achieved typically by approximating the integral by quadratures:

$$\int k(x, y)u(x)dx = \sum_{j=1}^N k(x_j, y)u(x_j)w_j$$

Discretizing the domain of the variable y , the equation can be written as:

$$\sum_{j=1}^N k(x_j, y_i)u(x_j)w_j + au(y_i) = f(y_i), \quad i = 1, \dots, N$$

If the kernel $k(x, y)$ is separable (or degenerate), then

$$k(x, y) = \sum_{l=1}^p \psi_l(x)\phi_l(y) + \epsilon(p)$$

where $\epsilon(p)$ is the truncation error bounded by p . Substituting in the discretized form of the integral equation

$$\sum_{j=1}^N u_j w_j \sum_{l=1}^p \psi_l(x_j)\phi_l(y_i) + au_i \cong f_i, \quad i = 1, \dots, N$$

Changing the order of summation results in:

$$\sum_{l=1}^p \phi_l(y_i) \sum_{j=1}^N \psi_l(x_j)w_j u_j + au_i = f_i, \quad i = 1, \dots, N$$

which require $O(Np + Np) = O(Np)$, $p \ll N$ operations per iteration as opposed to $O(N^2)$ operations for the conventional method.

5 Fast Fourier Transform and FMM

The Fast Fourier Transform (FFT) and the Fast Multipole Method (FMM) are similar in that they are both algorithms for fast matrix vector multiplication. In the following, some basics about both algorithms are compared.

- The Discrete Fourier Transform

The Fourier Transform of a function $h(t)$ is given by $H(f)$ where f is the frequency:

$$H(f) = \int_{-\infty}^{\infty} h(t)e^{2\pi ift} dt$$

$$h(t) = \int_{-\infty}^{\infty} H(f)e^{-2\pi ift} df$$

up to a constant normalizing term. If the function is sampled at discrete times, we get the Discrete Fourier Transform.

$$h(k) \equiv h(t_k), \quad t_k = k\Delta, \quad k = 0, 1, \dots, N-1$$

$$H(f_n) = \int_{-\infty}^{\infty} h(t)e^{2\pi if_n t} dt \approx \sum_{k=0}^{N-1} h_k e^{2\pi if_n t_k} \Delta = \Delta \sum_{k=0}^{N-1} h_k e^{2\pi i k n / N}$$

This ends up being a multiplication of a matrix (the Fourier Matrix) by a vector (sampled data).

$$H_n \equiv \sum_{k=0}^{N-1} h_k e^{2\pi i k n / N}$$

Note that the matrix H_n does not depend upon the time at all and has

- Dense Structured matrices

Both the FFT and FMM operate on dense but structured matrices.

- A *dense matrix* has a majority of non-zero elements. This is opposed to a *sparse* matrix.
- A *structured matrix* of order $N \times N$ is a dense matrix whose entries depend on only $O(N)$ parameters. Fast algorithms have been found for many structured matrices.

For FMM, matrices entries are derived from particular functions $\phi(y_j - x_i)$ that can be factorized (have their variables separated). For FFT, the Fourier matrix of order n is defined as:

$$F_n = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \dots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix}$$

where $\omega_n = e^{-\frac{2\pi i}{n}}$, is an n^{th} root of unity.

Note each ω_n is a fixed complex number (does not depend on time).

- Key idea

- In FMM, the key idea is to expand the function ϕ that derives the matrix entries, separate its variables (factorization), then change the order of summation.

- In FFT, the basic idea is to split the summation of order N into two summations of order $N/2$, and proceed recursively until we get “summations” of length 1. This is done by the use of Danielson

Lanczos lemma:

$$\begin{aligned}
F_k &= \sum_{j=0}^{N-1} e^{2\pi ijk/N} f_j \\
&= \sum_{j=0}^{N/2-1} e^{2\pi ik(2j)/N} f_{2j} + \sum_{j=0}^{N/2-1} e^{2\pi ik(2j+1)/N} f_{2j+1} \\
&= \sum_{j=0}^{N/2-1} e^{2\pi ikj/(N/2)} f_{2j} + \omega^k \sum_{j=0}^{N/2-1} e^{2\pi ikj/(N/2)} f_{2j+1} \\
&= F_k^e + \omega^k F_k^o
\end{aligned}$$

So, the Discrete Fourier Transform (DFT) of order N has been expressed as a sum of two DFTs of order $N/2$. This is faster since $(N/2)^2 + (N/2)^2 = N^2/2 < N^2$. This is repeated recursively:

$$F_k^e = F_k^{ee} + \omega^k F_k^{eo}, \quad F_k^o = F_k^{oe} + \omega^k F_k^{oo}$$

so as to reach eventually one point transformations which are equivalent to identity.

· Complexity

– The full fledged FMM requires $O(N \log N)$ operations for evaluating the product of an $N \times N$ matrix by an N vector.

– The FFT requires also $O(N \log N)$ operations. As shown above, each F_k is a sum of $\log_2 N$ transforms, and there are N such F_k 's.

· Approximation

– The FFT is exact (up to machine precision)

– The FMM is approximate. There exists a truncation error in addition to round-off error.