Fast Multipole Methods: Fundamentals & Applications

Ramani Duraiswami
Nail A. Gumerov

Week 1. Introduction.

- What are multipole methods and what is this course about.
- Problems from physics, mathematics, computer vision, statistics, etc. that can be solved with fast multipole methods.
- Historical review of the fast multipole method.
- Review of Complexity
- Homework.

Course Mechanics

- Background Needed
  - Linear Algebra
    - Matrices, Vectors, Linear Systems, LU Decomposition, Eigenvalue problem, SVD
  - Numerical Analysis
    - (Interpolation, Approximation, Finite Differences, Taylor Series, etc.)
  - Programming
    - Matlab, C/C++ and/or FORTRAN 9x
  - For particular applications in your domain you will need to know the background material there
  - E.g., familiarity with partial differential equations or integral equations, or interpolation, or …
- Participation essential!

Course Requirements

- Ideally
  - you are an applied student in engineering, physics or CS who has a problem in mind and would like to explore how the FMM could be applied to it
  - or, you are Applied Math/Scientific Computation student who wants to learn this important algorithm
- At the end of the course you will have a familiarity with
  - Application of FMM to different problems
  - Data structures for FMM, and analysis related to it.
- Course focus is on applying the methods to achieve solution.
  - Theory as needed to proceed

Homework

- Will try to have it every week
- Will not be excessive
- Essential for learning --- must do as opposed to just read.
- Homework handed out last class of a week.
- Due last class of next week
- Spring break week no homework

Projects & Exams

- There will be a final project that will require you to implement an FMM algorithm in a field of your choice,
  - account for 20% of the grade.
- Project to be chosen latest by March 13.
  - Implementation (reimplementation) of an FMM algorithm (hints and help will be given)
  - If you already have a project in mind you can discuss it with us
- Exams
  - intermediate exam worth 10%, week of March 11
  - final exam worth 20%. Finals Week

Class web & Mailing List

- http://www.umiacs.umd.edu/~ramani/cmsc878R
  - Discussions, announcements
- Homework will be posted on the web
  - No late homework (except by **timely** prior arrangement)
  - Hardcopy (no web or email submissions)
- Solutions will be posted after homework collected
- Links to papers etc.

Introductions

- Email addresses
- What are your interests?
- What do you want us to cover?
  - Last four years outline is posted at the course web site

Big Picture

- Object of all science
- Efficiency and better understanding by finding structure
  - More compact representation and ability to predict things
  - Ability to use structure for efficiency
- Types of structure
  - Geometric, Algebraic
  - Symmetry
  - Invariance to transformations
  - Compact representations
  - Approximation …
- Quite loose here with terminology

Structure in Numerical Methods

- Matrices
  - Diagonal, Banded, Sparse, Dense …
- Data
  - Uniformly sampled, normal, "low-dimensional", known with a given error, …
- Grids, Mesh
  - Cartesian, random
- Message: If you can identify structure, you can exploit it
- Becomes really important as problem size grows

Problem Sizes Continue to Grow in all Fields

- Sensors are getting varied and cheaper; and storage is getting cheaper
- Cameras, microphones
- Text (all the newspapers, books, technical papers)
- Genome data
- Medical/biological data (X-Ray, PET, MRI, Ultrasound, Electron microscopy …)
- Climate (Temperature, Salinity, Pressure, Wind, Oxygen content, …)
- Finer detail in meshes

We can do a lot with the data

- Build Models
- Learn trends
- Machine Learning
- Search …

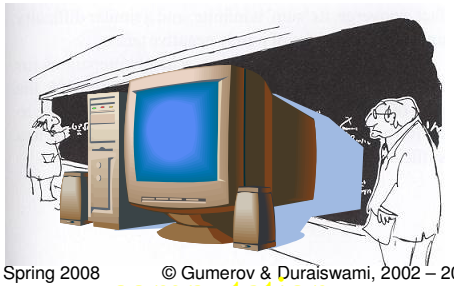
- New paradigm: data and computation driven science

- How about costs of doing these?
  - Faster computers?

math
algorithms

experimentation

Algorithm: graham scan
Find rightmost lowest point; label it p_0.
Sort all other points angularly about p_0.
In case of tie, delete the point closer to p_0
(or all but one copy for multiple points).
Stack S = (p_1, p_0)=(p_t, p_{t-1}); indexes top.
$i = 2$
while $i < n$ do
if p_i is strictly left of $p_{t-1}p_t$
then Push(p_i , S) and set $i \leftarrow i + 1$
else Pop(S).

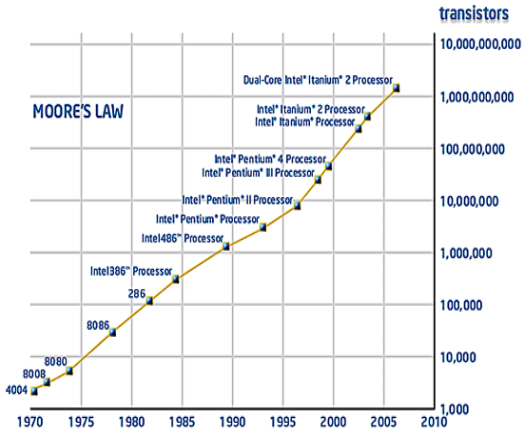


computation

Slide from Chazelle, 2006

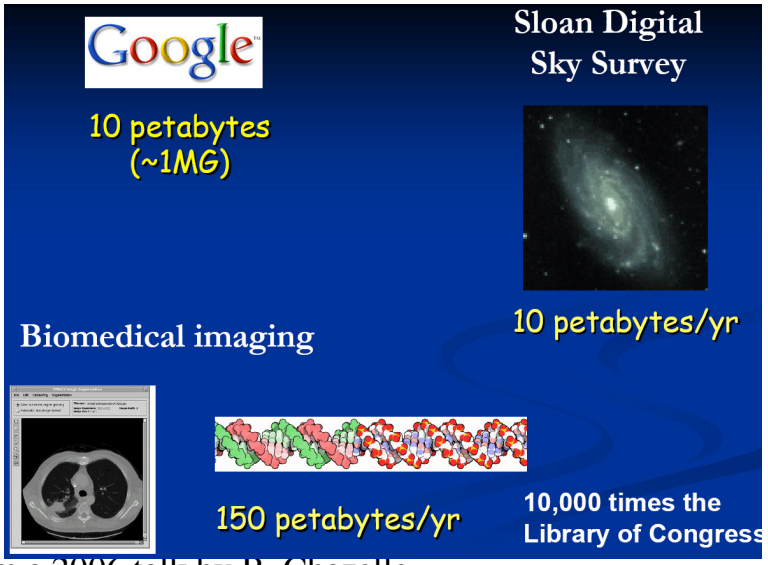
Moore's Law will take care of it

- Not law but an observation by Gordon Moore in the 1960s
- Number of transistors doubles every 18 months
- Basically means the "standard computer"s performance improves exponentially, with a doubling time of 18 months



Other exponential laws

- Internet data: doubling every year
- Camera megapixels about every 18 months
- Google's database started out on a 150 GB disk
  Currently estimates at ~100 petabytes
- Gates' law: The speed of commercial software generally slows by fifty percent every 18 months.
  (never formulated explicitly by Bill Gates, but rather relates to observations of Microsoft products).



- From a 2006 talk by B. Chazelle

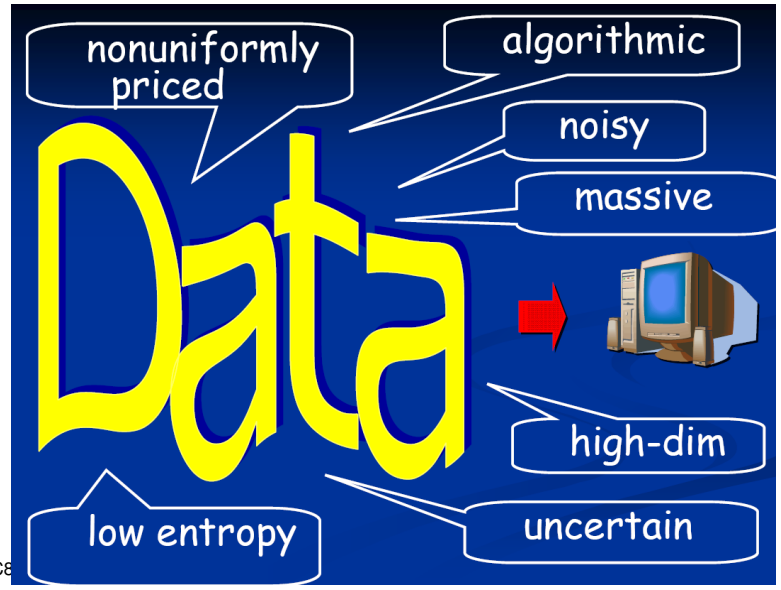
Refuting the Moore's law argument

- Argument:
  - Moore's law: Processor speed doubles every 18 months
  - If we wait long enough the computer will get fast enough and let my inefficient algorithm tackle the problem
- Is this true?
  - Yes for algorithms with same asymptotic complexity
  - No!! For algorithms with different asymptotic complexity
- For a million variables, we would need about 16 generations of Moore's law before a $O(N^2)$ algorithm was comparable with a $O(N)$ algorithm
- Similarly, clever problem formulation can also achieve large savings.

Algorithms

- Forget exponential or even polynomial time algorithms
- The future is
  - Linear or N log N algorithms.
  - Perhaps sublinear algorithms
  - Exploit structure in the data or prior knowledge
  - Approximation, randomization/sampling
  - Purpose built parallelization (e.g. graphical processors)

Structure in the data



Fast Fourier Transforms

- Allow matrix vector product (Fourier transform), and linear system solution (inverse Fourier transform) of data of size N in N log N time
- Require the data to be uniformly sampled

Top ten algorithms of the century [1]

1. Monte Carlo method.
2. Simplex method of linear programming.
3. Krylov Subspace Iteration method.
4. Householder matrix decomposition.
5. Fortran compiler.
6. QR algorithm for eigenvalue calculation.
7. Quicksort algorithm.
8. Fast Fourier Transform.
9. Integer Relation Detection Algorithm.
10. Fast Multipole methods.

[1] Dongarra, J. and Sullivan, F. 2000. The top ten algorithms of the century. Computing in Science and Engineering.

What is the Fast Multipole Method?

- An algorithm for achieving fast products of particular dense matrices with vectors
- Similar to the Fast Fourier Transform
  - For the FFT, matrix entries are uniformly sampled complex exponentials
- For FMM, matrix entries are
  - Derived from particular functions
  - Functions satisfy known "translation" theorems
- Name is a bit unfortunate
  - What is a multipole? We will return to this …
- Why is this important?

Vectors and Matrices

d dimensional column vector $\mathbf{x}$ and its transpose

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

- $n \times d$ dimensional matrix $\mathbf{M}$ and its transpose $\mathbf{M}^t$

$$\mathbf{M} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & \dots & m_{1d} \\ m_{21} & m_{22} & m_{23} & \dots & m_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & m_{n3} & \dots & m_{nd} \end{pmatrix} \text{ and}$$

$$\mathbf{M}^t = \begin{pmatrix} m_{11} & m_{21} & \dots & m_{n1} \\ m_{12} & m_{22} & \dots & m_{n2} \\ m_{13} & m_{23} & \dots & m_{n3} \\ \vdots & \vdots & \ddots & \vdots \\ m_{1d} & m_{2d} & \dots & m_{nd} \end{pmatrix}.$$

Matrix vector product

$$s_1 = m_{11} x_1 + m_{12} x_2 + \dots + m_{1d} x_d$$
$$s_2 = m_{21} x_1 + m_{22} x_2 + \dots + m_{2d} x_d$$
$$\dots$$
$$s_n = m_{n1} x_1 + m_{n2} x_2 + \dots + m_{nd} x_d$$

- Matrix vector product is identical to a sum

$$s_i = \sum_{j=1}^{d} m_{ij} x_j$$

- So algorithm for fast matrix vector products is also a fast summation algorithm

- d products and sums per line
- N lines
- Total Nd products and Nd sums to calculate N entries

Linear Systems

- Solve a system of equations

$$Mx=s$$

- M is a $N \times N$ matrix, x is a N vector, s is a N vector
- Direct solution (Gauss elimination, LU Decomposition, SVD, …) all need $O(N^3)$ operations
- Iterative methods typically converge in k steps with each step needing a matrix vector multiply $O(N^2)$
  - if properly designed, $k << N$
- A fast matrix vector multiplication algorithm ($O(N \log N)$ operations) will speed all these algorithms

Memory complexity

- Sometimes we are not able to fit a problem in available memory
  - Don't care how long solution takes, just if we can solve it
- To store a $N \times N$ matrix we need N^2 locations
  - 1 GB RAM = 1024^3 =1,073,741,824 bytes
  - => largest N is 32,768
- "Out of core" algorithms copy partial results to disk, and keep only necessary part of the matrix in memory
- FMM allows reduction of memory complexity as well
  - *Elements of the matrix required for the product can be generated as needed*

The need for fast algorithms

- Grand challenge problems in large numbers of variables
- Simulation of physical systems
  - Electromagnetics of complex systems
  - Stellar clusters
  - Turbulence
- Graphics and Vision
  - Light scattering …
- Molecular dynamics
  - General problems in these areas can be posed in terms of millions (10^6) or billions (10^9) of variables
  - Recall Avogadro's number (6.022 141 99 $\times$ 10^{23} molecules/mole
  - Protein folding

Machine Learning and Statistics

- Non-Parametric Modeling: "Let the data talk"
- Idea: let each data point participate in the density estimation/learning task
- Influence of each data point propagated to a neighborhood using a kernel function
- Methods are robust and quite successful
- Kernel density estimation/Parzen Windows in density estimation
- Kernel methods in learning
  - Regularized Least-Squares Classification
  - Gaussian Process Regression/Classification

Dense and Sparse matrices

- Operation estimates are for **dense matrices**.
  - Majority of elements of the matrix are ***non-zero***
- However in many applications matrices are ***sparse***
- A sparse matrix (or vector, or array) is one in which most of the elements are zero.
  - If storage space is more important than access speed, it may be preferable to store a sparse matrix as a list of (index, value) pairs.
  - For a given sparsity structure it may be possible to define a fast matrix-vector product/linear system algorithm

- Can compute

$$\begin{bmatrix} a_1 & 0 & 0 & 0 & 0 \\ 0 & a_2 & 0 & 0 & 0 \\ 0 & 0 & a_3 & 0 & 0 \\ 0 & 0 & 0 & a_4 & 0 \\ 0 & 0 & 0 & 0 & a_5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} a_1x_1 \\ a_2x_2 \\ a_3x_3 \\ a_4x_4 \\ a_5x_5 \end{bmatrix}$$

In 5 operations instead of 25 operations

- Sparse matrices are not our concern here

Structured matrices

- Fast algorithms have been found for many dense matrices
- Typically the matrices have some *"structure"*
- Definition:
  - A dense matrix of order $N \times N$ is called structured if its entries depend on only $O(N)$ parameters.
- Most famous example – the fast Fourier transform

Fourier Matrices

A Fourier matrix of order n is defined as the following

$$F_n = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{bmatrix},$$

where

$$\omega_n = e^{-\frac{2\pi i}{n}},$$

is an nth root of unity.

