Fast Multipole and Related Algorithms

Ramani Duraiswami
University of Maryland, College Park

http://www.umiacs.umd.edu/~ramani

Joint work with Nail A. Gumerov
Efficiency by exploiting symmetry and structure

• A general paradigm
  – Develop theory, algorithms and software that apply to general classes of data
  – Recognize underlying mathematical symmetry
  – Recognize structure in data
  – Adapt algorithms to particular data

• How does this apply to the FMM?
FMM

• Fast summation of singular kernel functions
  – Green’s functions $\Phi(x,y)$ for classical operator $L$
  – Solution $y$ to forcing at points $x$

• Applications
  – Electrostatics, Molecular/Stellar dynamics, Vortex
  – Boundary Integral Methods
  – Particle discretization of field equations

$$s(x_j) = \sum_{i=1}^{N} \alpha_i \phi(x_j - x_i), \quad \{s_j\} = [\Phi_{ji}]\{\alpha_i\}.$$
What is the FMM?

- Decompose singular sum into
  - “local” or sparse part +
  - “far-field” and dense part

\[
\phi(y_j) = \sum_{i=1}^{N} q_i \Phi(y_j - x_i), \quad j = 1, 2, \ldots, M
\]

\[
x_i, y_j \in \mathbb{R}^d.
\]

\[
\phi(y_j) = \sum_{x_i \notin \Omega(y_j)} q_i \Phi(y_j - x_i) + \sum_{x_i \in \Omega(y_j)} q_i \Phi(y_j - x_i)
\]
Factorization Trick – apply to dense part

• Not FMM, but has some key ideas

• Consider

\[ v(y_j) = \sum_{i=1}^{N} \Phi_{ij} u_i = \sum_{i=1}^{N} u_i (x_i - y_j)^2 \quad j=1, \ldots, M \]

• Naïve way to evaluate sum requires \( MN \) operations

• Instead can write the sum as

\[ v(y_j) = (\sum_{i=1}^{N} u_i) y_j^2 + (\sum_{i=1}^{N} u_i x_i^2) - 2y_j (\sum_{i=1}^{N} u_i x_i) \]

  – Can evaluate each bracketed sum over \( i \)

\[ \beta = (\sum_{i=1}^{N} u_i), \quad \gamma = (\sum_{i=1}^{N} u_i x_i^2), \quad \delta = (\sum_{i=1}^{N} u_i x_i) \]

and evaluate

\[ v(y_j) = \beta y_j^2 + \gamma - 2y_j \delta \]

  – Requires \( O(M+N) \) operations
Reduction of Complexity

Straightforward (nested loops):

\[
\begin{align*}
&\text{for } j = 1, \ldots, M \\
&\quad v_j = 0; \\
&\quad \text{for } i = 1, \ldots, N \\
&\quad \quad v_j = v_j + \Phi(y_j, x_i) u_i; \\
&\quad \quad \text{end;} \\
&\quad \text{end;} \\
\end{align*}
\]

Complexity: \(O(MN)\)

Factorized:

\[
\begin{align*}
&\text{for } m = 0, \ldots, p - 1 \\
&\quad c_m = 0; \\
&\quad \text{for } i = 1, \ldots, N \\
&\quad \quad c_m = c_m + a_m(x_i - x_*) u_i; \\
&\quad \quad \text{end;} \\
&\quad \text{end;} \\
&\text{for } j = 1, \ldots, M \\
&\quad v_j = 0; \\
&\quad \text{for } m = 0, \ldots, p - 1 \\
&\quad \quad v_j = v_j + c_m f_m(y_j - x_*) \\
&\quad \quad \text{end;} \\
&\quad \text{end;} \\
\end{align*}
\]

Complexity: \(O(pN+pM)\)

If \(p \ll \min(M,N)\) then complexity reduces!
• Take this far-near decomposition and apply it recursively
  – Far field and near field
  – Take near-field and divide further
  – Use translations
Data Structures

• Separate data into local and global
• Manage the factorizations
• Basic geometric construction --- WSPD

• Stuff inside one area, expanded as a series
• Translate series representation to other area
• Expand in the other area
Data Structures

- Need to automatically structure the source and target point sets
- Can’t tile space automatically with non-overlapping circles/spheres
- Classically, use quadtree/octrees
- A recent paper
Box and its neighbors

$E_1$, $E_2$, $E_3$, $E_4$
Data structures must support

• *Separation*: Cells satisfy local separation and WSPD properties across all levels.

• *Indexing*: Cell indices satisfy some order relation in memory.

• *Spatial addressing*: Cell centers are computable from addresses and each particle can find its bounding cell.

• *Hierarchical addressing*: Cell children, parent, and neighbor indices are computable from addresses.

• All this must be done in $O(N \log N)$ and must be parallelizable


Direct method vs. FMM

Straightforward

$O(NM)$

MLFMM

Source Data Hierarchy
Evaluation Data Hierarchy

$O(N+M)$
Translational

- Representation size \( p_1 \) needs to be translated to another representation of size \( p_2 \)
- In general this is a \( O(p_1 p_2) \) computation
- Can reduce this cost by looking at symmetries and approximations of the matrix
  - Coaxial and rotational symmetries
  - Exponential expansions and Integral Relations
  - SVD based approximations

Helmholtz equation

• Issue: size of special function based factorizations grow with wavenumber times distance
• Means FMM is actually slower than direct multiplication at high frequencies!
• Rokhlin solution: diagonal factorization
• Our solution: level dependent truncation number; use of rotation based symmetries and local expansion with special functions; switch to diagonal representations later
• More numerically stable; easier to implement in BEM; cheaper
FMM on the GPU

- GPUs are extremely fast at doing computations which have high arithmetic intensity for each load/store, esp. MADD
- Do other computations, but not with as much speed-up

A few cores

Hundreds cores

NVIDIA Tesla C2050:
- 1.25 Tflops single
- 0.52 Tflops double
- 448 cores
GPU/heterogeneous FMM

- Moved to implement FMM on GPU almost with the introduction of CUDA (2007)
- FMM translation based on rotation translation, and showing that it is competitive or superior to asymptotically faster diagonal translations
- Real valued formulations
- Algorithm splitting
- Extension to heterogeneous architectures by mapping local sums to GPU (where they are most efficient) and tree based FMM to CPU
Local summation on GPU (FMM)

CPU:
Time = CNs, s = 8^{-l_{\text{max}}} N

GPU:
Time = A_1 N + B_1 N/s + C_1 Ns

read/write
float computations
access to box data

These parameters depend on the hardware

FMM on heterogeneous architectures

• Insight --- why do gymnastics to fit tree algorithms with irregular access on the GPU

• GPUs hosted usually with a multicore CPU

• Let the GPU do local sum, rest on the CPU
Task 3.5: Computational Considerations in Brownout Simulations

Single node algorithm

**GPU work**
- particle positions, source strength
- data structure (octree and neighbors)

**CPU work**
- ODE solver: source receiver update
- source $S$-expansions
- translation stencils
- local direct sum
- upward $S|S$
- downward $S|R$
- receiver $R$-expansions
- final sum of far-field and near-field interactions

**Time loop**
Task 3.5: Computational Considerations in Brownout Simulations

The algorithm flow chart

- The algorithm flow chart illustrates the process of solving for positions and source strength, data structure (octree), merge octree, data structure (neighbors), redistributed particle, single heterogeneous node algorithm, exchange final R-expansions, and final sum.

Node A:
- ODE solver: source receiver update
- Positions, source strength
- Data structure (octree)
- Merge octree
- Data structure (neighbors)
- Redistributed particle
- Single heterogeneous node algorithm
- Exchange final R-expansions
- Final sum

Node B:
- ODE solver: source receiver update
- Positions, source strength
- Data structure (octree)
- Merge octree
- Data structure (neighbors)
- Redistributed particle
- Single heterogeneous node algorithm
- Exchange final R-expansions
- Final sum
Fast Gauss Transforms in high dimensions

• Sums of Gaussians arise in many applications in many dimensions
• Vortex methods; heat kernels; statistical density estimation; kernel methods in machine learning
• FMM data structures and factorizations do not work in higher dimensions
• Alternate global factorization; Data structures (based on k-center algorithm); and a automatically tuned algorithm developed in a series of papers
• Downloadable code from SourceForge (FigTree)
• Many papers applying this to vision, machine learning etc.

A few pretty pictures from papers
Teaching the FMM

- Course notes since 2004 online
- www.umiacs.umd.edu/~ramani/cmsc878R
- Java Applet that demos the FMM
- 8 papers directly from the course
- enabled solidifying broad themes