Fast N-body Algorithms for Dynamic Problems on the GPU

Qi Hu, Nail A. Gumerov and Ramani Duraiswami, University of Maryland, College Park
{huqi, gumerov, ramani}@umiacs.umd.edu

Fast Multipole Methods

• Approximate evaluation of sum of potentials (or radial basis functions)

• Functions can be in 1,2,3 or d dimensions

• Reduce cost from $O(N^2)$ to $O(N (\log N + \log \varepsilon))$ for given error ε

• Many application areas including f

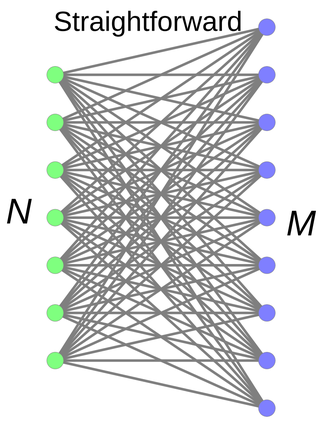
- Astrophysics, Molecular dynamics
- Scattering calculations (Acoustics, EM)
- Partial Differential Equations
- Statistics, Machine Learning

$$\phi(\mathbf{y}_j) = \sum_{i=1}^{N} q_i \Phi(\mathbf{y}_j, \mathbf{x}_i), \quad \nabla\phi(\mathbf{y}_j) = \sum_{i=1}^{N} q_i \nabla\Phi(\mathbf{y}_j, \mathbf{x}_i), \quad j = 1,\ldots,M,$$

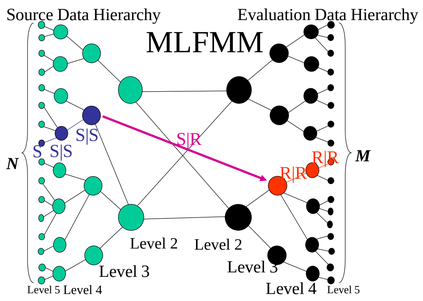
E.g. Laplace kernel in 3D: $\Phi(\mathbf{y}_j, \mathbf{x}_i) = |\mathbf{y}_j - \mathbf{x}_i|^{-1}$.

FMM: A hierarchical algorithm

•Uses data structures (octrees, neighbor lists) and translation theorems to reduce $O(N^2)$ cost

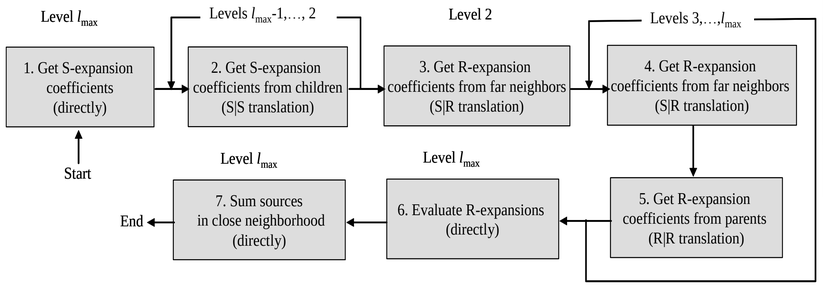


O(NM)



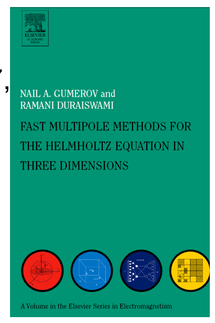
O(N+M)

More formal flowchart



References:

1. L. Greengard & V. Rokhlin, “A fast algorithm for particle simulations,” J. Comput. Phys., 73, 1987, 325-348.
2. N.A. Gumerov & R. Duraiswami. *Fast Multipole Methods for the Helmholtz Equation in Three Dimensions*, Elsevier, 2005.
3. N.A. Gumerov & R. Duraiswami. *Fast Multipole Methods on Graphics Processors*, J. Comput. Phys.,227, 2008, 8290-8313
4. *M.S. Warren, J.K. Salmon, D.J. Becker, M.P. Goda, T. Sterling, & G.S. Winckelmans. “Pentium Pro inside: I. a treecode at 430 Gigaflops on ASCI Red,” Bell price winning paper at SC’97, 1997*



Previous FMM realized on GPU/CPU

• The adaption of FMM to the GPU was presented by Gumerov & Duraiswami [3]
• Multi Language Environment: Fortran 95 on CPU, CUDA on GPU, Middleware FLAGON as a bridge
• Data structures were generated on CPU and transferred to GPU
• The FMM run was performed on the GPU
• However expensive CPU data structure construction makes the implementation unsuitable for computing dynamic problems

FMM Data Structures on GPU

• Pack source & receiver data points according to their max-level box indices in O(n) time (one pass) on GPU. One point per thread:
  • Using CUDA atomic add operation to count
  • Adapting scan operation in CUDA SDK to build boxes bookmark for data points addressing
  • Using bookmark to filter empty boxes

• Build neighbor list and neighbor bookmark to address the data points for the direct sum. One thread per non-empty receiver box at the max level:
  • Double link from neighbor bookmark to neighbor list then to sorted source data points enable us to avoid empty box storage
  • Adapting scan to build neighbor bookmark

• Data structures for translation are build on the fly:
  • For each box of current level, build its non-empty status and compute its E2 or E4 neighbors. One box per thread.
  • Each thread load pre-computed translation coefficients for each non-empty box and compute S|S, S|R or R|R translation.

• All data structures are computed and stored on GPU without data transfer between host and device

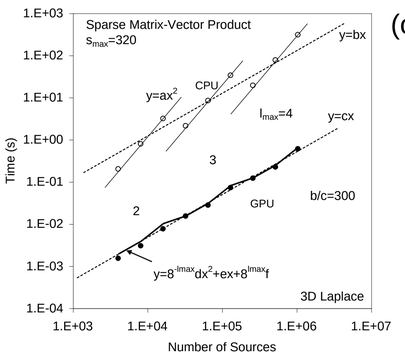
GPU Data Structure Performance test

GPU: NVIDIA GeForce 480 GTX
CPU: Intel Xeon "Nehalem“ Quad-Core 2.8Ghz

1048576 Source & Receiver Points			
Level	CPU Time	GPU Time	Speedup Ratio
3	223,42	7.686	29.068
4	272.254	13.949	19.517
5	430.616	12.959	33.229
6	1808.414	34.591	52.280
7	6789.339	70.847	95.831
8	7782.755	124.859	62.332

Performance tests

The FMM computation results are from [3].
CPU: 2.67 GHz Intel Core 2 extreme QX 7400 (2GB RAM and one of four CPUs employed).
GPU: NVIDIA GeForce 8800 GTX (peak 330 GFLOPS).



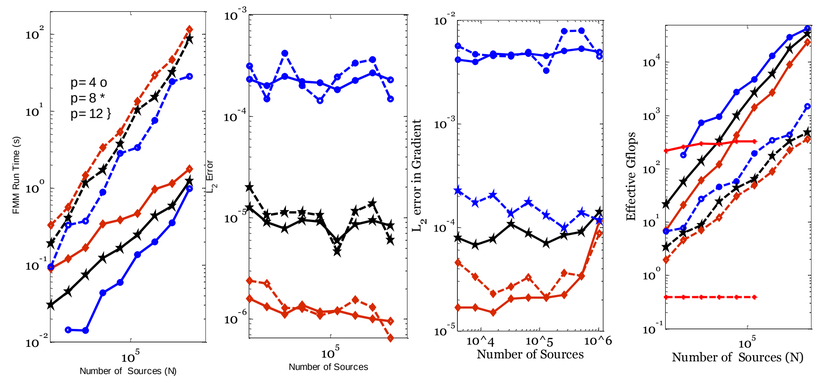
Sparse mat-vec product

(optimal settings for CPU and GPU)

N	Serial CPU (s)	GPU(s)	Time Ratio
4096	3.51E-02	1.50E-03	23
8192	1.34E-01	2.96E-03	45
16384	5.26E-01	7.50E-03	70
32768	3.22E-01	1.51E-02	21
65536	1.23E+00	2.75E-02	45
131072	4.81E+00	7.13E-02	68
262144	2.75E+00	1.20E-01	23
524288	1.05E+01	2.21E-01	47
1048576	4.10E+01	5.96E-01	69

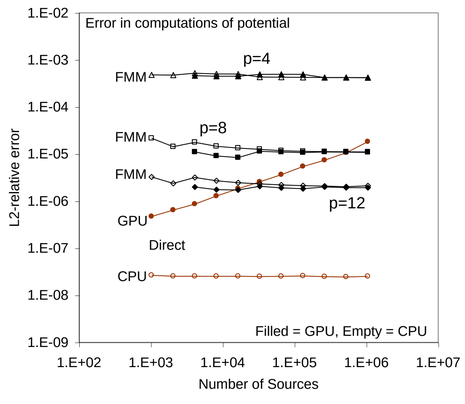
Other steps of the FMM
• Straigtforward accelerations in range 5-60;
• Effective accelerations for $N=10^6$ (taking into account octree level reduction) in range 30-60;
• Remark: Direct summation (no FMM) can be accelerated up to 600 times.

Overall Performance



Results achieved on our FMM code on the CPU and GPU for N=4096,…, 1048576. ***Left:*** Wall-clock time for the FMM run portion for different accuracy settings on the CPU and GPU. A roughly linear scaling is observed. Also shown are the direct product results for the GPU and CPU (the lines with slope =2). ***Left Middle:*** Error achieved on the CPU and GPU versions of the FMM for different truncation numbers. ***Right Middle:*** Error achieved in force calculation on the CPU and GPU versions of the FMM for different truncation numbers. ***Right:*** Effective number of Gflops achieved (following [3]). While the peak performance of the direct potential/force evaluation is about 290 GFlops, the FMM achieves speeds of up to 50 Tflops.

Error in computation of large sums on GPU using direct summation and the FMM



Error computed over a grid of 729 sampling points, relative to “exact” solution, which is direct summation with double precision.

Possible reason why the GPU error in direct summation grows: systematic roundoff error in computation of function 1/sqrt(x). (still a question).