Parallel FMM

Matthew Knepley

Computation Institute
University of Chicago

Department of Mathematics and Computer Sciences
Széchenyi István University, Győr, Hungary
October 1, 2010

RUSH UNIVERSITY
MEDICAL CENTER

Main Point

Using estimates and proofs,

a simple software architecture,

gets good scaling, efficiency,

and adaptive load balance.

Main Point

Using estimates and proofs,

a simple software architecture,

gets good scaling, efficiency,

and adaptive load balance.

Main Point

Using estimates and proofs,

a simple software architecture,

gets good scaling, efficiency,

and adaptive load balance.

Collaborators

The PetFMM team:

- Prof. Lorena Barba
  - Dept. of Mechanical Engineering, Boston University
- Dr. Felipe Cruz, developer of GPU extension
  - Nagasaki Advanced Computing Center, Nagasaki University
- Dr. Rio Yokota, developer of 3D extension
  - Dept. of Mechanical Engineering, Boston University

Collaborators

Chicago Automated Scientific Computing Group:

- Prof. Ridgway Scott
  - Dept. of Computer Science, University of Chicago
  - Dept. of Mathematics, University of Chicago
- Peter Brune, (biological DFT)
  - Dept. of Computer Science, University of Chicago
- Dr. Andy Terrel, (Rheagen)
  - Dept. of Computer Science and TACC, University of Texas at Austin

Outline

FMM Work

- Queue-based hybrid execution
  - OpenMP for multicore processors
  - CUDA for GPUs
- Adaptive hybrid Treecode-FMM
  - Treecode competitive only for very low accuracy
  - Very high flop rates for treecode M2P operation
- Computation/Communication Overlap FMM
  - Provably scalable formulation
  - Overlap P2P with M2L

Other Work

- Classical DFT in Biology
  - Excellent speedup over CPU
  - Enabled 3D simulations of calcium ion channels
- PetRBF: radial basis functions on the GPU
  - 10-20x speedup over CPU
  - Combined with PetFMM for full vortex fluid method code
- FEM: Autogenerated optimized kernels
  - Autogenerate code for hundreds of elements, and generic weak forms using FEniCS
  - Achieve 20% of peak for 3D P_1 elements (10x over CPU)

Outline

FMM Applications

FMM can accelerate both integral and boundary element methods for:

- Laplace
- Stokes
- Elasticity

FMM Applications

FMM can accelerate both integral and boundary element methods for:

- Laplace
- Stokes
- Elasticity

Advantages

- Mesh-free
- $\mathcal{O}(N)$ time
- Distributed and multicore (GPU) parallelism
- Small memory bandwidth requirement

Fast Multipole Method

FMM accelerates the calculation of the function:

$$\Phi(x_i) = \sum_j K(x_i, x_j) q(x_j) \tag{1}$$

- Accelerates $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$ time
- The kernel $K(x_i, x_j)$ must decay quickly from (x_i, x_i)
  - Can be singular on the diagonal (Calderón-Zygmund operator)
- Discovered by Leslie Greengard and Vladimir Rohklin in 1987
- Very similar to recent wavelet techniques

Fast Multipole Method

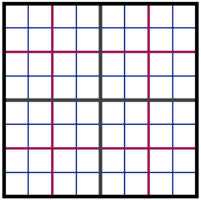
FMM accelerates the calculation of the function:

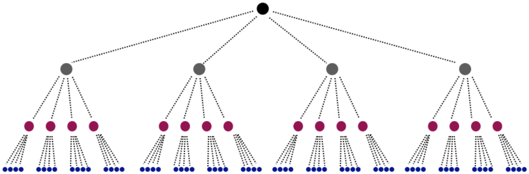
$$\Phi(x_i) = \sum_j \frac{q_j}{|x_i - x_j|} \qquad (1)$$

- Accelerates $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$ time
- The kernel $K(x_i, x_j)$ must decay quickly from (x_i, x_i)
  - Can be singular on the diagonal (Calderón-Zygmund operator)
- Discovered by Leslie Greengard and Vladimir Rohklin in 1987
- Very similar to recent wavelet techniques

Spatial Decomposition

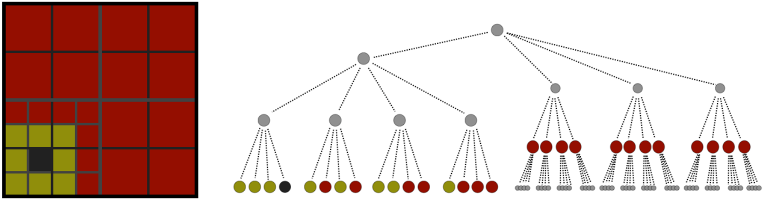
Pairs of boxes are divided into *near* and *far*:





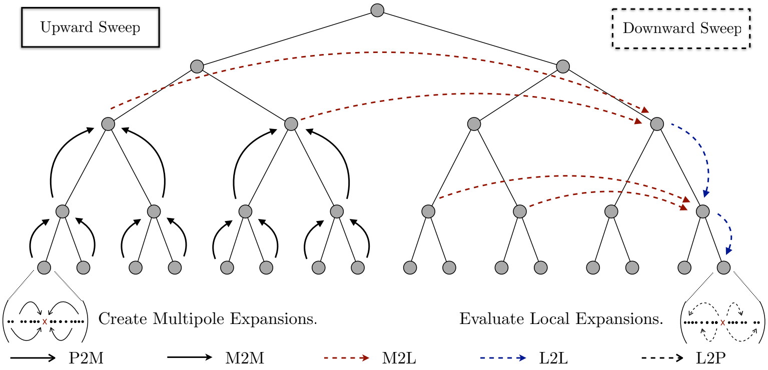
Spatial Decomposition

Pairs of boxes are divided into *near* and *far*:



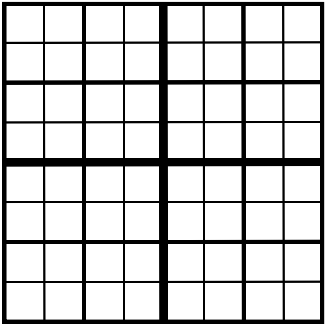
Neighbors are treated as *very near*.

Functional Decomposition



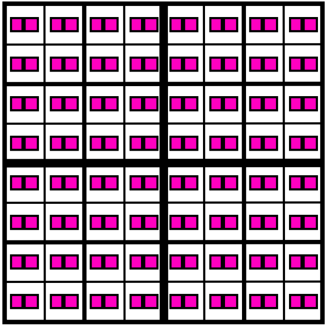
Outline

FMM in Sieve



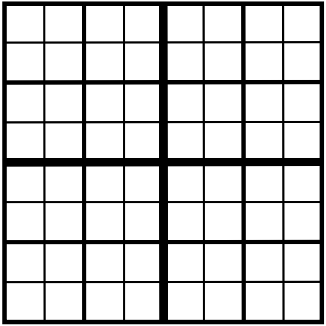
- The Quadtree is a `Sieve`
  - with optimized operations
- Multipoles are stored in `Sections`
- Two `Overlaps` are defined
  - Neighbors
  - Interaction List
- Completion moves data for
  - Neighbors
  - Interaction List

FMM in Sieve



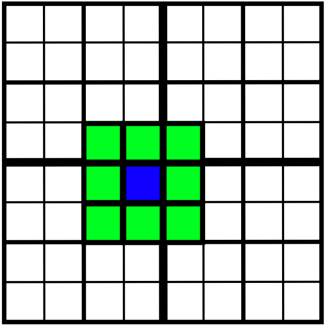
- The Quadtree is a `Sieve`
  - with optimized operations
- Multipoles are stored in `Sections`
- Two `Overlaps` are defined
  - Neighbors
  - Interaction List
- Completion moves data for
  - Neighbors
  - Interaction List

FMM in Sieve



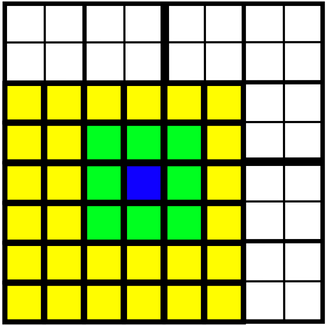
- The Quadtree is a `Sieve`
  - with optimized operations
- Multipoles are stored in `Sections`
- Two `Overlaps` are defined
  - Neighbors
  - Interaction List
- Completion moves data for
  - Neighbors
  - Interaction List

FMM in Sieve



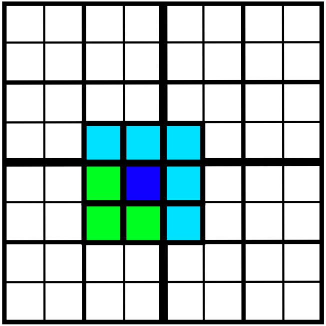
- The Quadtree is a `Sieve`
  - with optimized operations
- Multipoles are stored in `Sections`
- Two `Overlaps` are defined
  - Neighbors
  - Interaction List
- Completion moves data for
  - Neighbors
  - Interaction List

FMM in Sieve



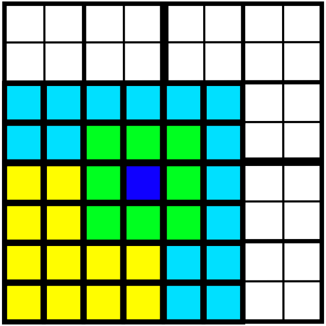
- The Quadtree is a `Sieve`
  - with optimized operations
- Multipoles are stored in `Sections`
- Two `Overlaps` are defined
  - Neighbors
  - Interaction List
- Completion moves data for
  - Neighbors
  - Interaction List

FMM in Sieve



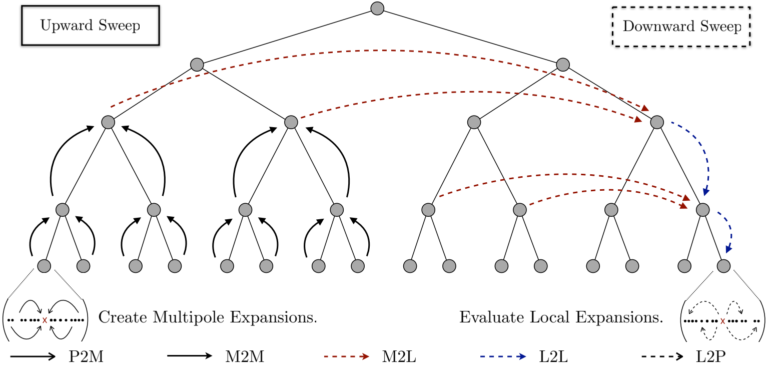
- The Quadtree is a `Sieve`
  - with optimized operations
- Multipoles are stored in `Sections`
- Two `Overlaps` are defined
  - Neighbors
  - Interaction List
- Completion moves data for
  - Neighbors
  - Interaction List

FMM in Sieve



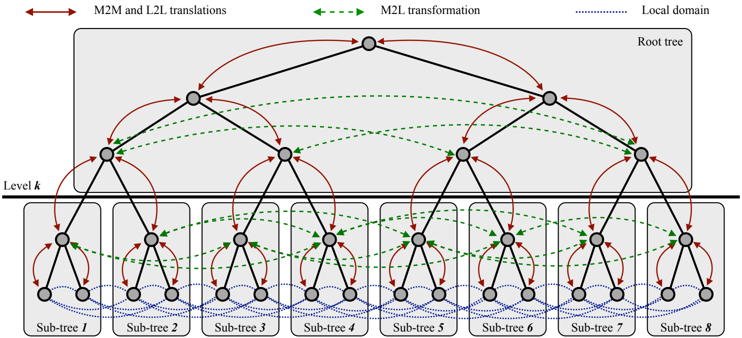
- The Quadtree is a `Sieve`
  - with optimized operations
- Multipoles are stored in `Sections`
- Two `Overlaps` are defined
  - Neighbors
  - Interaction List
- Completion moves data for
  - Neighbors
  - Interaction List

FMM Control Flow



Kernel operations will map to GPU tasks.

FMM Control Flow
Parallel Operation



Kernel operations will map to GPU tasks.

Parallel Tree Implementation

- Divide tree into a root and local trees
- Distribute local trees among processes
- Provide communication pattern for local sections (overlap)
  - Both neighbor and interaction list overlaps
  - Sieve generates MPI from high level description

Parallel Tree Implementation

How should we distribute trees?

- Multiple local trees per process allows good load balance
- Partition weighted graph
  - Minimize load imbalance and communication
  - Computation estimate:

Leaf	$N_i p$ (P2M) + $n_I p^2$ (M2L) + $N_i p$ (L2P) + $3^d N_i^2$ (P2P)
Interior	$n_c p^2$ (M2M) + $n_I p^2$ (M2L) + $n_c p^2$ (L2L)

  - Communication estimate:

Diagonal	$n_c(L - k - 1)$
Lateral	$2^d \frac{2^{m(L-k-1)}-1}{2^m-1}$ for incidence dimesion m

- Leverage existing work on graph partitioning
  - ParMetis

Parallel Tree Implementation
Why should a good partition exist?

Shang-hua Teng, Provably good partitioning and load balancing algorithms for parallel adaptive N-body simulation, SIAM J. Sci. Comput., **19**(2), 1998.

- Good partitions exist for non-uniform distributions
  - 2D $\mathcal{O}\left(\sqrt{n}(\log n)^{3/2}\right)$ edgecut
  - 3D $\mathcal{O}\left(n^{2/3}(\log n)^{4/3}\right)$ edgecut
- As scalable as regular grids
- As efficient as uniform distributions
- ParMetis will find a nearly optimal partition

Parallel Tree Implementation

Will ParMetis find it?

George Karypis and Vipin Kumar, Analysis of Multilevel Graph Partitioning, Supercomputing, 1995.

- Good partitions exist for non-uniform distributions
  - 2D $C_i = 1.24^i C_0$ for random matching
  - 3D $C_i = 1.21^i C_0$?? for random matching
- 3D proof needs assurance that averge degree does not increase
- Efficient in practice

Parallel Tree Implementation
Advantages

- Simplicity
- Complete serial code reuse
- Provably good performance and scalability

Parallel Tree Implementation
Advantages

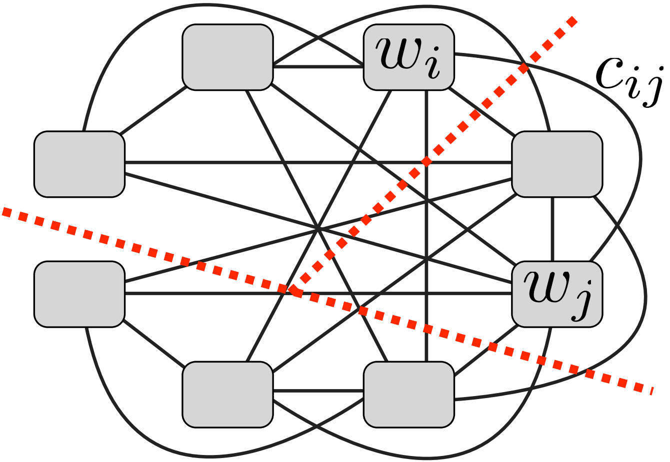
- Simplicity
- Complete serial code reuse
- Provably good performance and scalability

Parallel Tree Implementation
Advantages

- Simplicity
- Complete serial code reuse
- Provably good performance and scalability

Distributing Local Trees

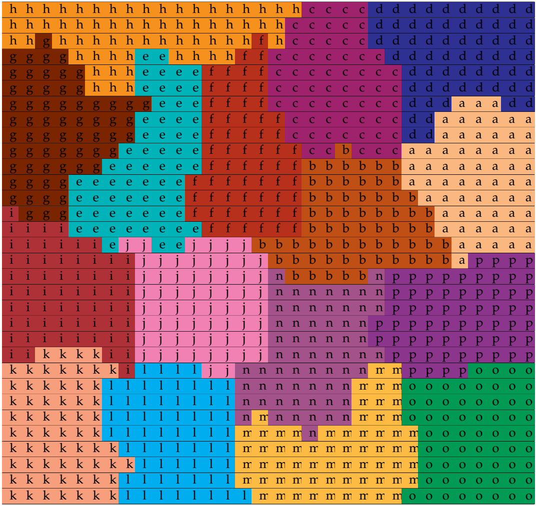
The interaction of locals trees is represented by a weighted graph.



This graph is partitioned, and trees assigned to processes.

Local Tree Distribution

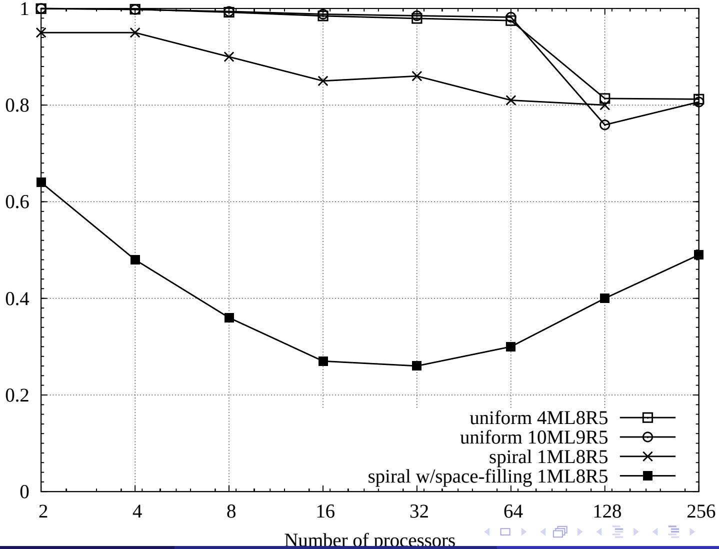
Here local trees are assigned to processes:



Parallel Data Movement

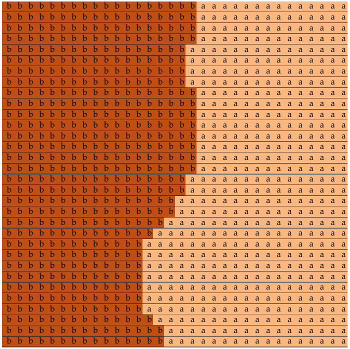
1. Complete neighbor section
2. Upward sweep
    1. Upward sweep on local trees
    2. Gather to root tree
    3. Upward sweep on root tree
3. Complete interaction list section
4. Downward sweep
    1. Downward sweep on root tree
    2. Scatter to local trees
    3. Downward sweep on local trees

PetFMM Load Balance

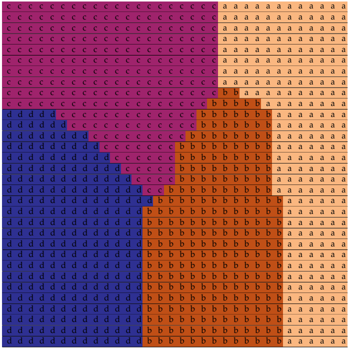


Local Tree Distribution

Here local trees are assigned to processes for a spiral distribution:



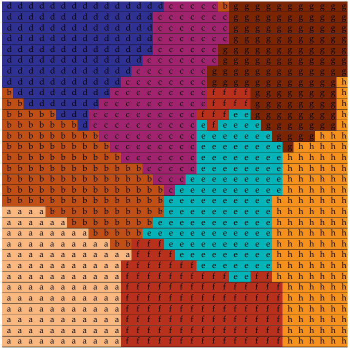
(a) 2 cores



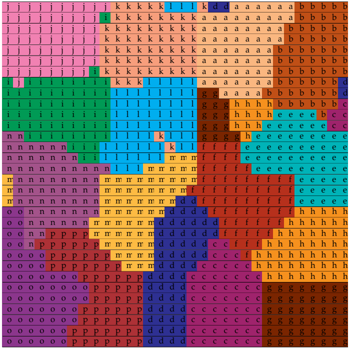
(b) 4 cores

Local Tree Distribution

Here local trees are assigned to processes for a spiral distribution:



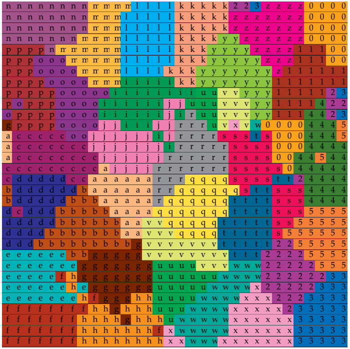
(c) 8 cores



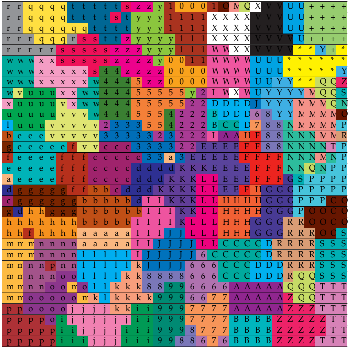
(d) 16 cores

Local Tree Distribution

Here local trees are assigned to processes for a spiral distribution:



(e) 32 cores



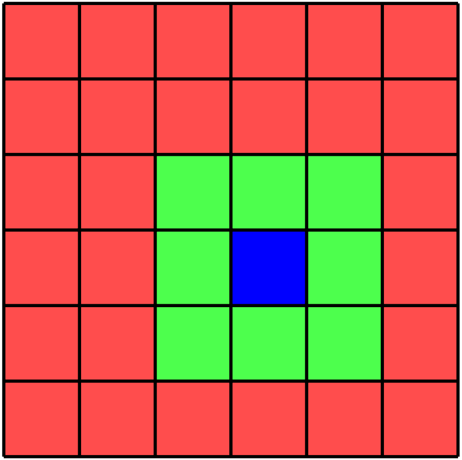
(f) 64 cores

Outline

Multipole-to-Local Transformation

Re-expands a multipole series as a Taylor series

- Up to 85% of time in FMM
  - Tradeoff with direct interaction
- Dense matrix multiplication
  - $2p^2$ rows
- Each interaction list box
  - $\left(6^d - 3^d\right) 2^{dL}$
- $d = 2, L = 8$
  - 1,769,472 matvecs



GPU M2L
Version 0

One thread per M2L transform

- Thread block (TB) transforms one Multipole Expansion (ME) for each Interaction List (IL) box — 27 times
- $p = 12$
- Matrix size is 2304 bytes
- Plenty of work per thread (81 Kflops or 36 flops/byte)
- **BUT**, 16K shared memory only holds 7 matrices

GPU M2L

Version 0

One thread per M2L transform

- Thread block (TB) transforms one Multipole Expansion (ME) for each Interaction List (IL) box — 27 times
- $p = 12$
- Matrix size is 2304 bytes
- Plenty of work per thread (81 Kflops or 36 flops/byte)
- **BUT**, 16K shared memory only holds 7 matrices

GPU M2L
Version 0

One thread per M2L transform

- Thread block (TB) transforms one Multipole Expansion (ME) for each Interaction List (IL) box — 27 times
- $p = 12$
- Matrix size is 2304 bytes
- Plenty of work per thread (81 Kflops or 36 flops/byte)
- **BUT**, 16K shared memory only holds 7 matrices

GPU M2L
Version 0

One thread per M2L transform

- Thread block (TB) transforms one Multipole Expansion (ME) for each Interaction List (IL) box — 27 times
- $p = 12$
- Matrix size is 2304 bytes
- Plenty of work per thread (81 Kflops or 36 flops/byte)
- **BUT**, 16K shared memory only holds 7 matrices

GPU M2L

Version 0

One thread per M2L transform

- Thread block (TB) transforms one Multipole Expansion (ME) for each Interaction List (IL) box — 27 times
- $p = 12$
- Matrix size is 2304 bytes
- Plenty of work per thread (81 Kflops or 36 flops/byte)
- **BUT**, 16K shared memory only holds 7 matrices

GPU M2L

Version 0

One thread per M2L transform

- Thread block (TB) transforms one Multipole Expansion (ME) for each Interaction List (IL) box — 27 times
- $p = 12$
- Matrix size is 2304 bytes
- Plenty of work per thread (81 Kflops or 36 flops/byte)
- **BUT**, 16K shared memory only holds 7 matrices

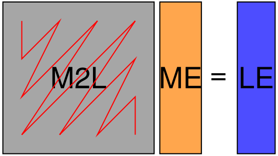
Memory limits concurrency!

GPU M2L
Version 1

Apply M2L transform matrix-free

$$\mathrm{m2l}_{ij} = -1^{i} \binom{i+j}{j} t^{-i-j-1} \quad (2)$$

- Traverse matrix by perdiagonals
- Same work
- No memory limit on concurrency
- 8 concurrent TBs per MultiProcessor (MP)
- $27 \times 8 = 216$ threads, **BUT** max is 512

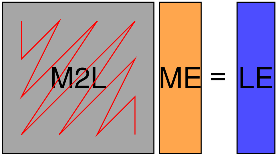


GPU M2L
Version 1

Apply M2L transform matrix-free

$$\mathrm{m2l}_{ij} = -1^{i} \binom{i+j}{j} t^{-i-j-1} \qquad (2)$$

- Traverse matrix by perdiagonals
- Same work
- No memory limit on concurrency
- 8 concurrent TBs per MultiProcessor (MP)
- $27 \times 8 = 216$ threads, **BUT** max is 512

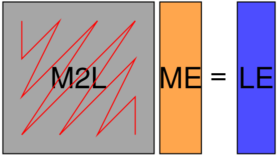


GPU M2L
Version 1

Apply M2L transform matrix-free

$$\mathrm{m2l}_{ij} = -1^i \binom{i+j}{j} t^{-i-j-1} \tag{2}$$

- Traverse matrix by perdiagonals
- Same work
- No memory limit on concurrency
- 8 concurrent TBs per MultiProcessor (MP)
- $27 \times 8 = 216$ threads, **BUT** max is 512



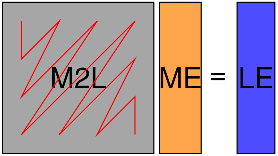
GPU M2L

Version 1

Apply M2L transform matrix-free

$$\mathrm{m2l}_{ij} = -1^{i} \binom{i+j}{j} t^{-i-j-1} \quad (2)$$

- Traverse matrix by perdiagonals
- Same work
- No memory limit on concurrency
- 8 concurrent TBs per MultiProcessor (MP)
- $27 \times 8 = 216$ threads, **BUT** max is 512



GPU M2L

Version 1

Apply M2L transform matrix-free

$$\mathrm{m2l}_{ij} = -1^{i} \binom{i+j}{j} t^{-i-j-1} \qquad (2)$$

- Traverse matrix by perdiagonals
- Same work
- No memory limit on concurrency
- 8 concurrent TBs per MultiProcessor (MP)
- $27 \times 8 = 216$ threads, **BUT** max is 512

20 GFlops

5x Speedup of Downward Sweep

GPU M2L
Version 1

Apply M2L transform matrix-free

$$\mathrm{m2l}_{ij} = -1^{i} \binom{i+j}{j} t^{-i-j-1} \tag{2}$$

- Traverse matrix by perdiagonals
- Same work
- No memory limit on concurrency
- 8 concurrent TBs per MultiProcessor (MP)
- $27 \times 8 = 216$ threads, **BUT** max is 512

20 GFlops

5x Speedup of Downward Sweep

Algorithm limits concurrency!

GPU M2L
Version 1

Apply M2L transform matrix-free

$$\mathrm{m2l}_{ij} = -1^i \binom{i+j}{j} t^{-i-j-1} \tag{2}$$

Additional problems: Not enough parallelism for data movement

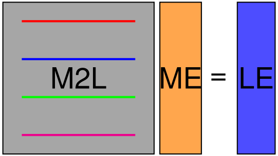
- Move 27 LE to global memory per TB
- $27 \times 2p = 648$ floats
- With 32 threads, takes 21 memory transactions

GPU M2L
Version 2

One thread per *element* of the LE

$$\mathrm{m2l}_{ij} = -1^{i} \binom{i+j}{j} t^{-i-j-1} \tag{3}$$

- Each thread does a dot product
- Cannot use diagonal traversal, more work
- Avoid branching
  - Each row precomputes t^{-i-1}
  - **All** threads loop to $p+1$, only **store** t^{-i-1}
- Loop unrolling
- No thread synchronization

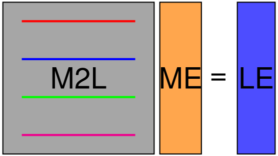


GPU M2L
Version 2

One thread per *element* of the LE

$$\mathrm{m2l}_{ij} = -1^{i} \binom{i+j}{j} t^{-i-j-1} \tag{3}$$

- Each thread does a dot product
- Cannot use diagonal traversal, more work
- Avoid branching
  - Each row precomputes t^{-i-1}
  - **All** threads loop to $p+1$, only **store** t^{-i-1}
- Loop unrolling
- No thread synchronization

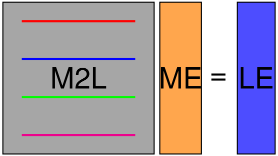


GPU M2L
Version 2

One thread per *element* of the LE

$$\mathrm{m2l}_{ij} = -1^{i} \binom{i+j}{j} t^{-i-j-1} \quad (3)$$

- Each thread does a dot product
- Cannot use diagonal traversal, more work
- Avoid branching
  - Each row precomputes t^{-i-1}
  - **All** threads loop to $p+1$, only **store** t^{-i-1}
- Loop unrolling
- No thread synchronization

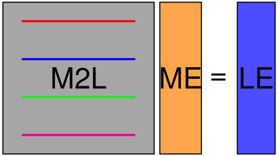


GPU M2L
Version 2

One thread per *element* of the LE

$$\mathrm{m2l}_{ij} = -1^{i} \binom{i+j}{j} t^{-i-j-1} \qquad (3)$$

- Each thread does a dot product
- Cannot use diagonal traversal, more work
- Avoid branching
  - Each row precomputes t^{-i-1}
  - **All** threads loop to $p+1$, only **store** t^{-i-1}
- Loop unrolling
- No thread synchronization



GPU M2L
Version 2

One thread per *element* of the LE

$$\mathrm{m2l}_{ij} = -1^{i} \binom{i+j}{j} t^{-i-j-1} \tag{3}$$

- Each thread does a dot product
- Cannot use diagonal traversal, more work
- Avoid branching
  - Each row precomputes t^{-i-1}
  - **All** threads loop to $p+1$, only **store** t^{-i-1}
- Loop unrolling
- No thread synchronization

300 GFlops

15x Speedup of Downward Sweep

GPU M2L

Version 2

One thread per *element* of the LE

$$\mathrm{m2l}_{ij} = -1^{i} \binom{i+j}{j} t^{-i-j-1} \tag{3}$$

- Each thread does a dot product
- Cannot use diagonal traversal, more work
- Avoid branching
  - Each row precomputes t^{-i-1}
  - **All** threads loop to $p+1$, only **store** t^{-i-1}
- Loop unrolling
- No thread synchronization

300 GFlops

15x Speedup of Downward Sweep

Examine memory access

Memory Bandwidth

Superior GPU memory bandwidth is due to both

bus width and clock speed.

	CPU	GPU
Bus Width (bits)	64	512
Bus Clock Speed (MHz)	400	1600
Memory Bandwidth (GB/s)	3	102
Latency (cycles)	240	600

Tesla always accesses blocks of 64 or 128 bytes

GPU M2L

Version 3

Coalesce and overlap memory accesses

Coalescing is

- a group of 16 threads
- accessing consective addresses
  - 4, 8, or 16 bytes
- in the same block of memory
  - 32, 64, or 128 bytes

GPU M2L
Version 3

Coalesce and overlap memory accesses

Memory accesses can be overlapped with computation when

- a TB is waiting for data from main memory
- another TB can be scheduled on the SM
- 512 TB can be active at once on Tesla

GPU M2L
Version 3

Coalesce and overlap memory accesses

Note that the theoretical peak (1 TF)

- MULT and FMA must execute simultaneously
- 346 GOps
- Without this, peak can be closer to 600 GF

480 GFlops

25x Speedup of Downward Sweep

Design Principles

M2L required all of these optimization steps:

- Many threads per kernel
- Avoid branching
- Unroll loops
- Coalesce memory accesses
- Overlap main memory access with computation

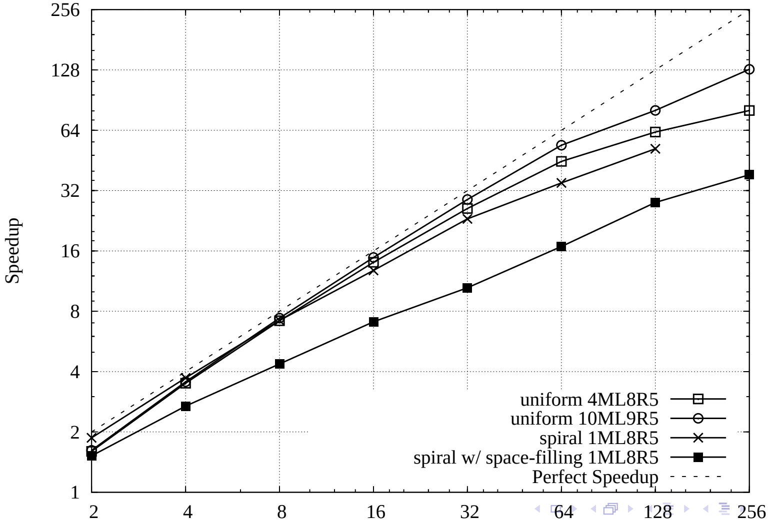
Outline

PetFMM

PetFMM is an freely available implementation of the
Fast Multipole Method
http://barbagroup.bu.edu/Barba_group/PetFMM.html

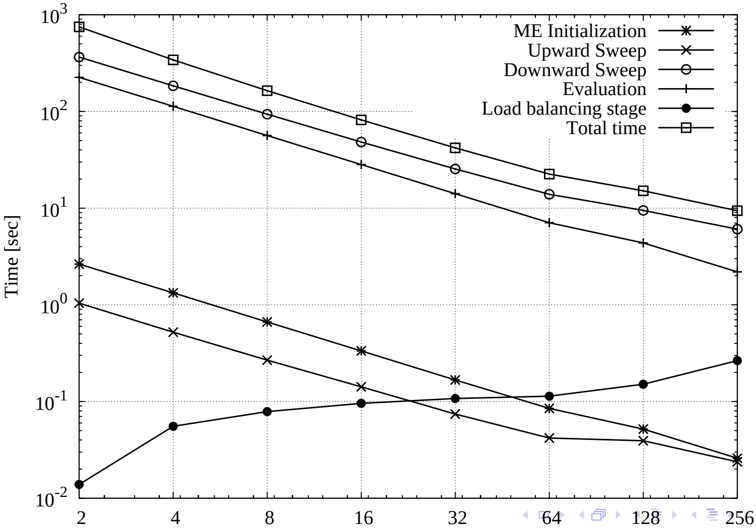
- Leverages PETSc
  - Same open source license
  - Uses Sieve for parallelism
- Extensible design in C++
  - Templated over the kernel
  - Templated over traversal for evaluation
- MPI implementation
  - Novel parallel strategy for anisotropic/sparse particle distributions
  - PetFMM–A dynamically load-balancing parallel fast multipole library
  - 86% efficient strong scaling on 64 procs
- Example application using the Vortex Method for fluids
- (coming soon) GPU implementation

PetFMM CPU Performance
Strong Scaling

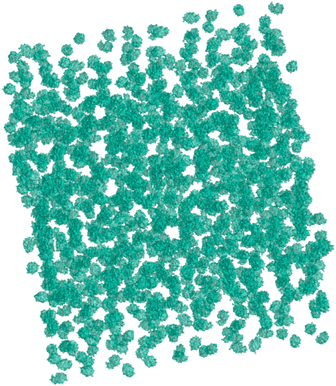


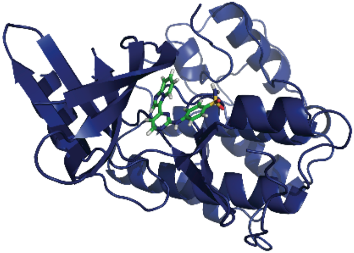
PetFMM CPU Performance

Strong Scaling



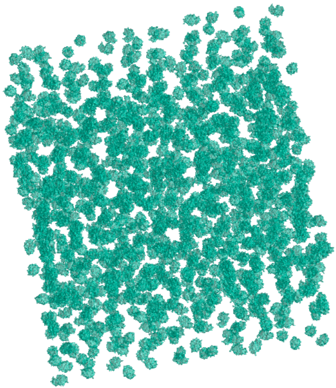
Largest Calculation With Development Code

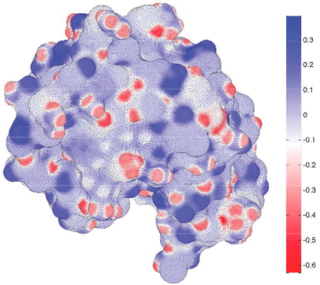




- 10,648 randomly oriented lysozyme molecules
- 102,486 boundary elements/molecule
- More than 1 billion unknowns
- 1 minute on 512 GPUs

Largest Calculation With Development Code





- 10,648 randomly oriented lysozyme molecules
- 102,486 boundary elements/molecule
- More than 1 billion unknowns
- 1 minute on 512 GPUs

How Will Algorithms Change?

- Massive concurrency is necessary
  - Mix of vector and thread paradigms
  - Demands new analysis
- More attention to memory management
  - Blocks will only get larger
  - Determinant of performance
- Urgent need for reduction in complexity
  - Complete serial code reuse
  - Modeling integral to optimization