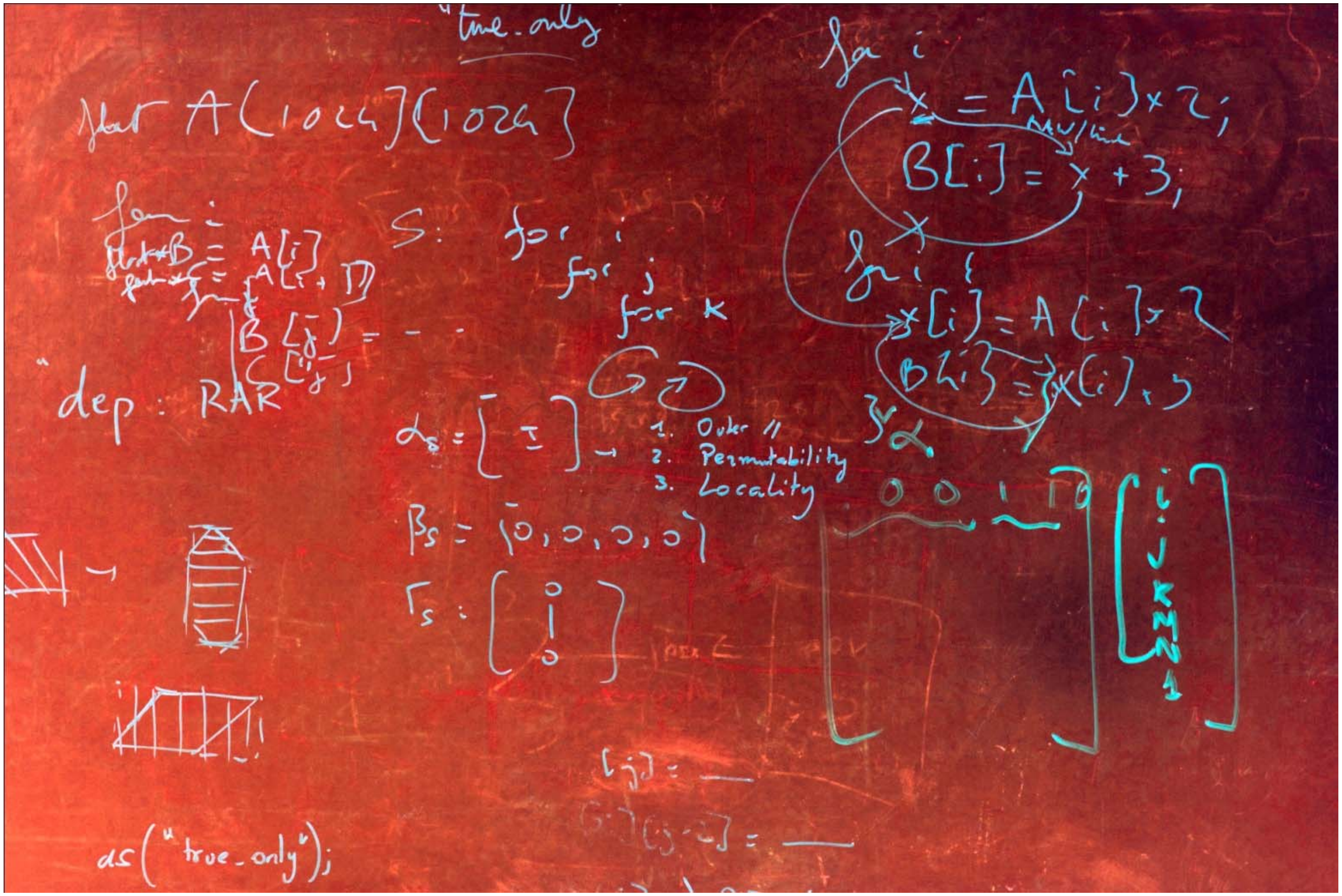
Re-Introduction of Communication-Avoiding FMM-Accelerated FFTs with GPU Acceleration

M. Harper Langston, Muthu Baskaran, Benoit Meister, Nicolas Vasilache and Richard Lethin, Reservoir Labs Inc.



Introduction and Motivation

FFTs are everywhere and continuous interest in improvements and reduced communication.

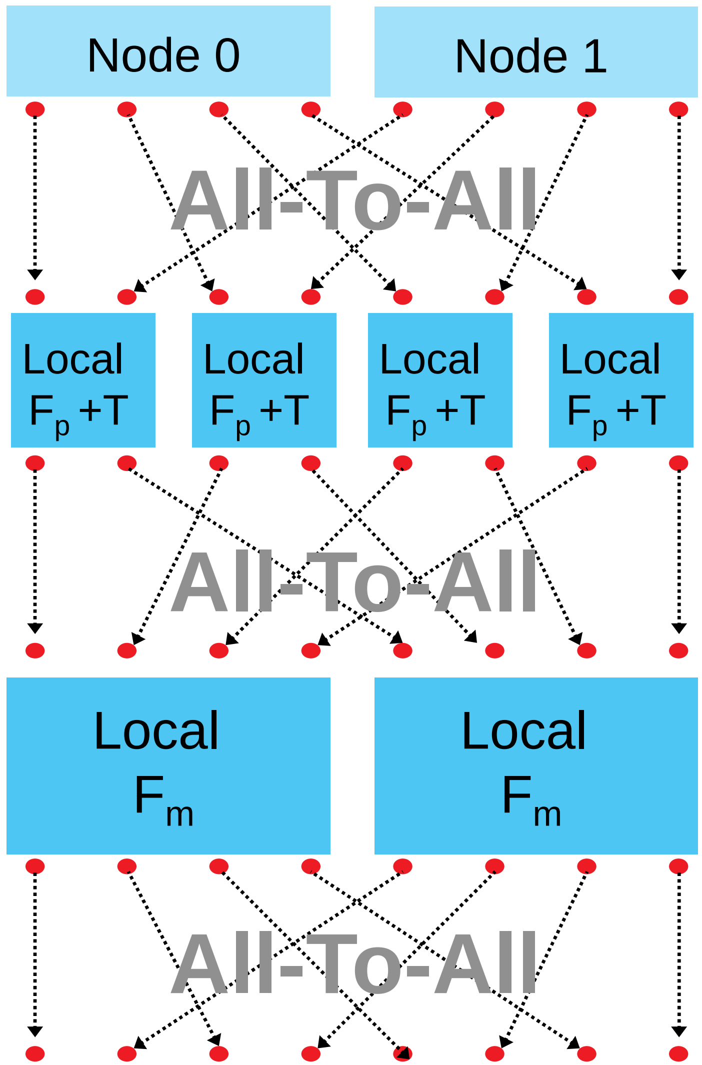
- sFFTs of interest for sparse frequency domains.
- For reducing communication from 3 global “all-to-all” calls to 1 for an in order, large 1D FFT, Tang et. al (SC’12) use an oversampling approach while mentioning older approach.
- Edelman et. al. (SIAM J.SciComp’97) has largely been ignored due to previous lack of interest and reliance on Fast Multipole Methods (FMMs).
- Our interest is in reinvestigating and working to optimize this older lower-communication 1D FMM-FFT.

Traditional In Order Parallel FFT Approach in 1D

Assume N = M*P Points and Apply The Operator:

- $F_n = (I_p \otimes F_m)(F_p \otimes I_m)\Pi$
- Perform Global Bit Reversal
- Perform Local FFTs and Global Transpose
- Apply Twiddle Factors
- Perform Local FFTs and Global Bit Reversal

- REQUIRES 3 GLOBAL ALL-TO-ALL CALLS!



Alternative Low-Communication FFTs

Reducing Global Communication from 3 to ~1

- Edelman et. al. (SIAM J.SciComp'97) refactor the operator as $F_n = (I_p \otimes F_m)(F_p \otimes I_m)M\Pi$ with factor matrices $M = diag(I_m, C^1, \ldots, C^{p-1})$,
  $C^s_{(j,k)} = \rho^s \left[cot\left(\frac{\pi}{m}(k - j + s/p)\right) + i\right]$,
  $\rho^s = exp(-i\pi s/p)sin(\pi s/p)/m$
- ***C*** matrices applied with FMM to reduce global communication:
  - In processor μ, evaluate $C^{\mu}\mathbf{x}_{\mu}$ in parallel with FMM (this incorporates the distributed $\Pi\mathbf{x}$ calculation);
  - Perform m = n/p p-sized distributed FFT operations, corresponding to $(F_p \otimes I_m)$;
  - For each processor μ, perform a local m-sized FFT, corresponding to $(I_p \otimes F_m)$

Why Use Fast Direct Solvers to Accelerate Refactorization?

Given a free-space or integral equation

$$u(x_i) = \sum_{j=1}^{n} G(x_i, y_j) f(y_j) = \sum_{j=1}^{n} G_{i,j} f_j$$

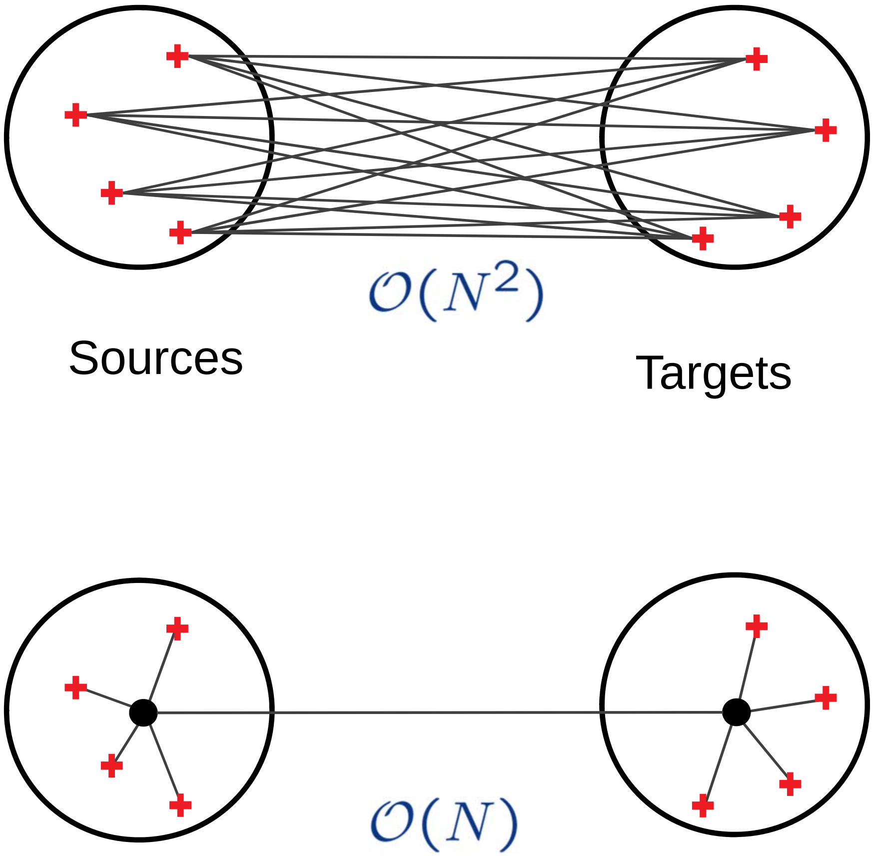
where G incorporates the kernel, K, and a quadrature weight, fast direct solvers seek to achieve the following desired properties:

- Ability to achieve desired accuracy;
- Increased computational efficiency;
- Suitable quadrature method.
- **FMM addresses all of these concerns.**

Fast Multipole Method: basic idea

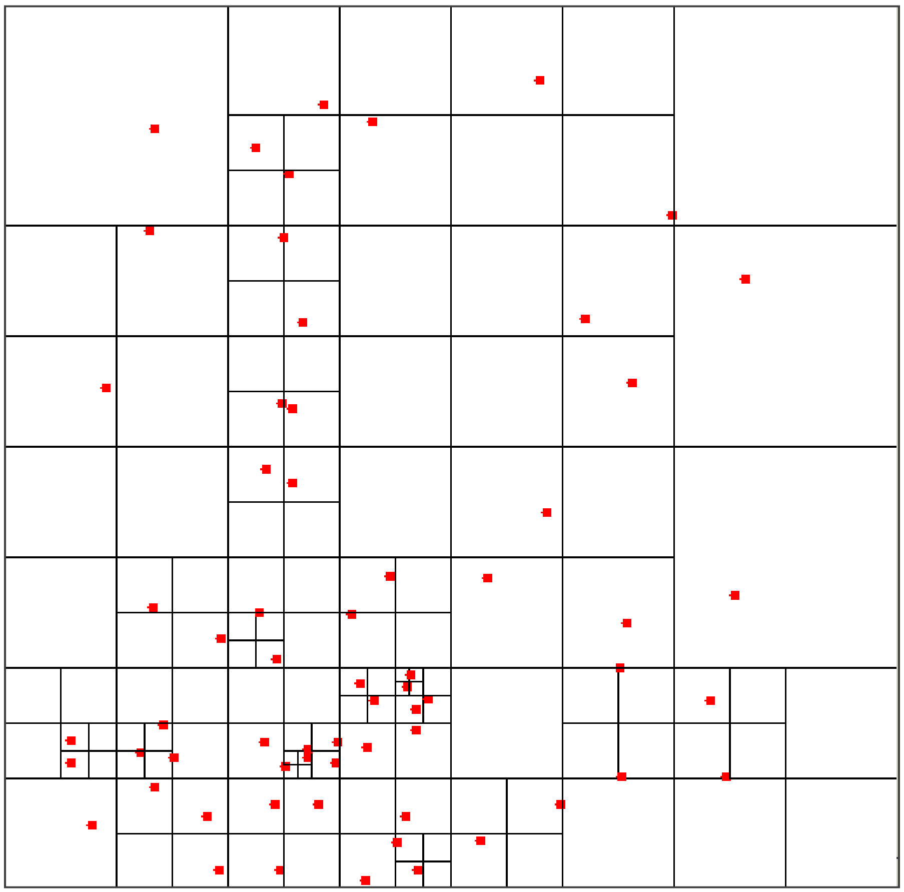
Original implementations
(Greengard, Rokhlin 1987):

- Subdivide space in an intelligent manner.
- Construct far and near fields.
- For all target locations outside of a circle of radius R, approximate potential from sources inside circle with a multipole expansion.
- For all target locations inside a circle of radius r, approximate potential from sources outside as a local expansion.



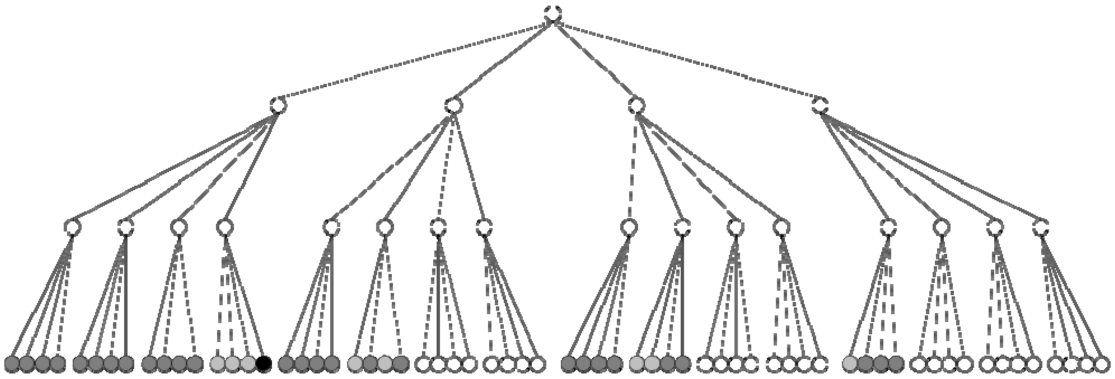
Fast Multipole Method

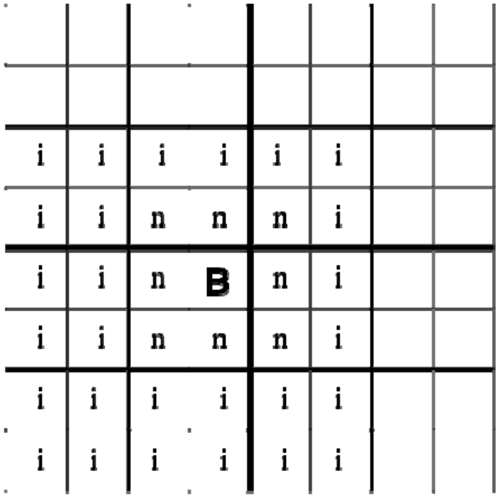
Tree Structure – 2D example



Fast Multipole Method

Tree Structure (Uniform/Nonadaptive)



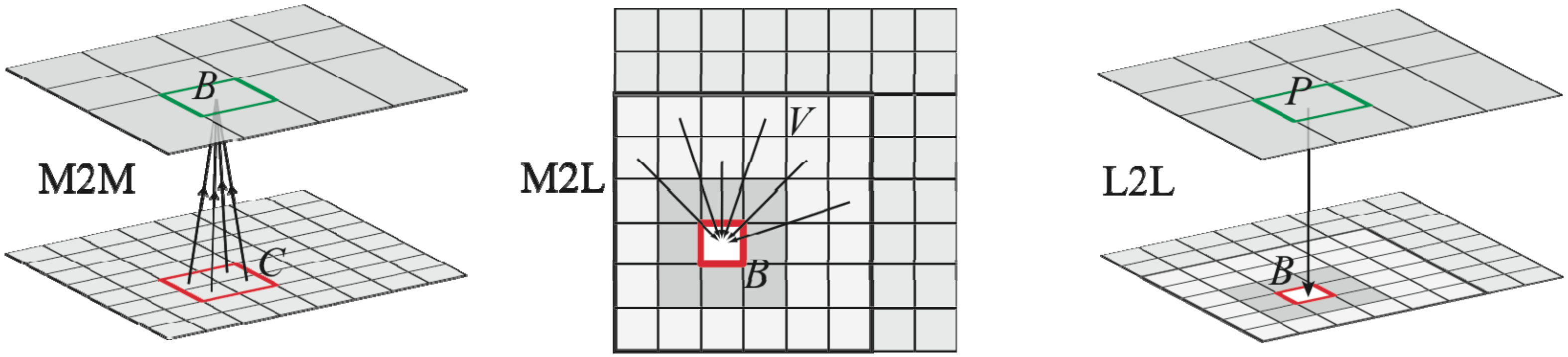


- Subdivided domain on right with marked interaction boxes and near neighbors.
- Tree data structure on left – leaves represent smallest.
- The Near-Field (NF) is the neighbor list and Far-Field (FF) is everything else.

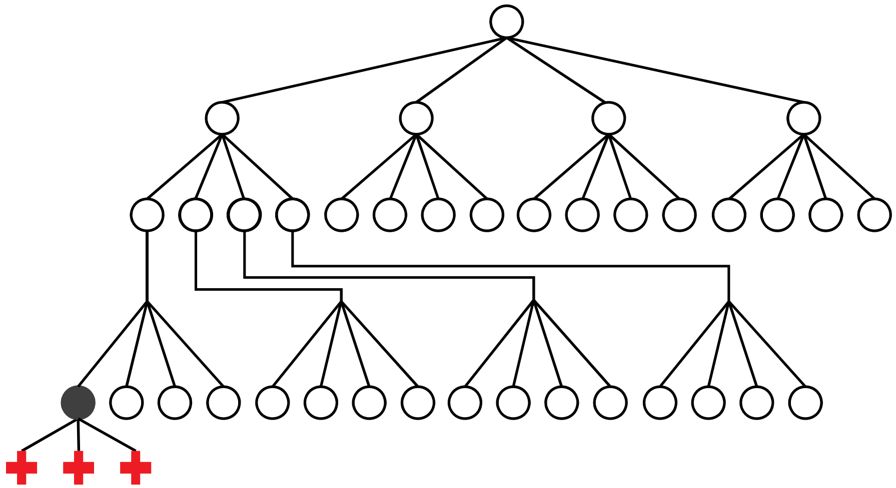
FMM Translation Operators and Steps Overview for Two Dimensional Illustrative Approach

Basic Steps

- Upward: M2M translation turns multipole expansions of box's children into its own multipole expansion.
- Downward: M2L translation turns multipole expansions of box into local expansion for non-adjacent box and L2L translation turns expansions of box's parent into local expansion for itself.
- Compute Near-Field (NF) Interactions at targets and L2T translates local expansions (FF or far-field interactions) to targets.

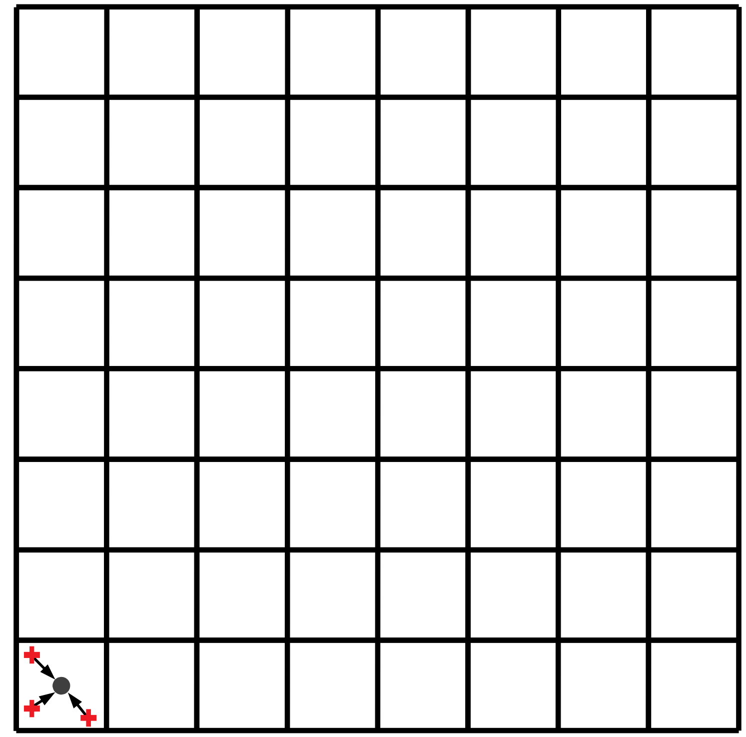


Upward pass

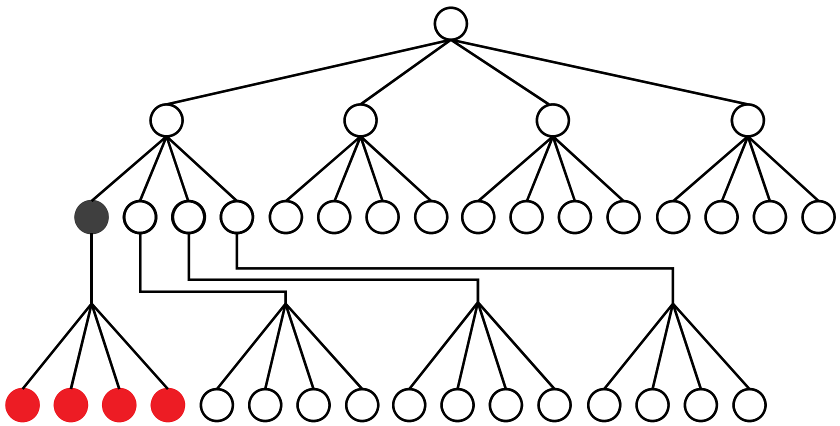


For each box, generate multipole expansions of its own sources

- If leaf, from exact source (S2M)
- If non-leaf, from children (M2M)

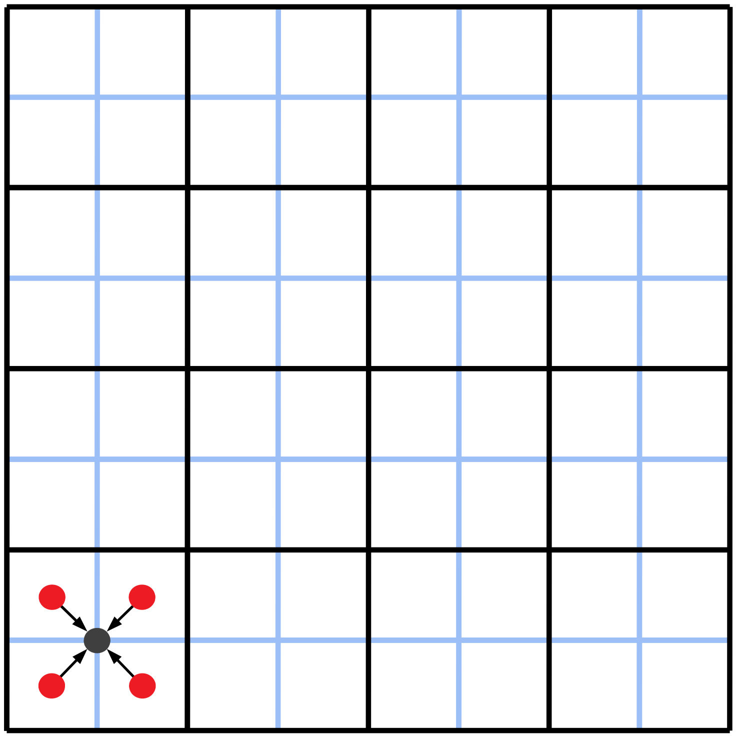


Upward pass



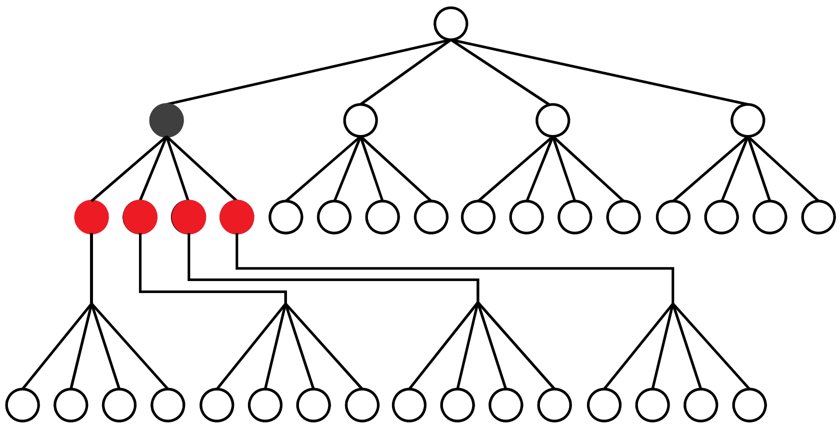
For each box, generate multipole expansions of its own sources

- If leaf, from exact source (S2M)
- If non-leaf, from children (M2M)



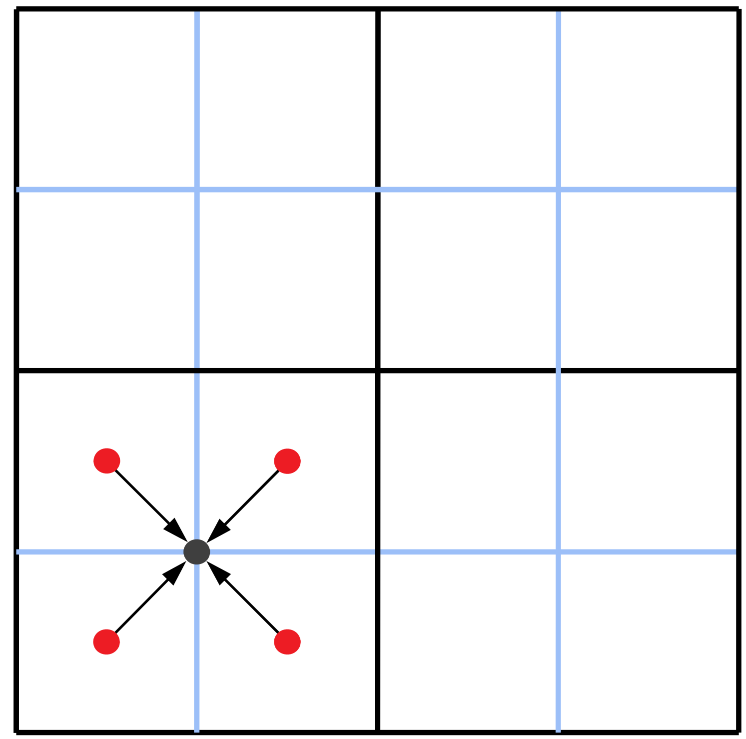
M2M

Upward pass



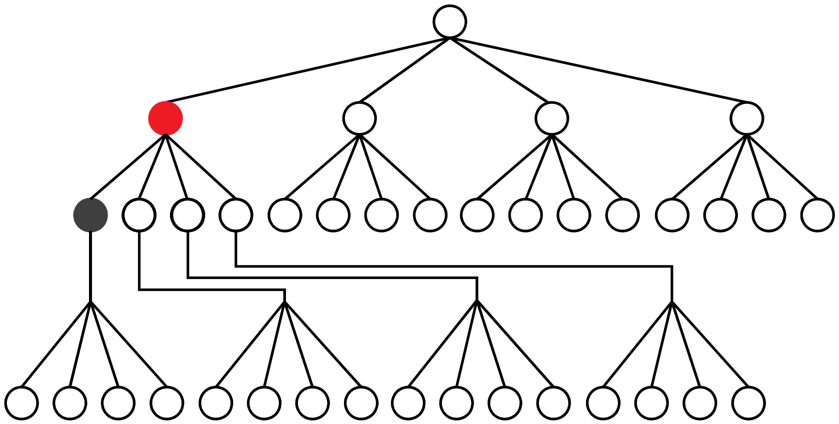
For each box, generate multipole expansions of its own sources

- If leaf, from exact source (S2M)
- If non-leaf, from children (M2M)



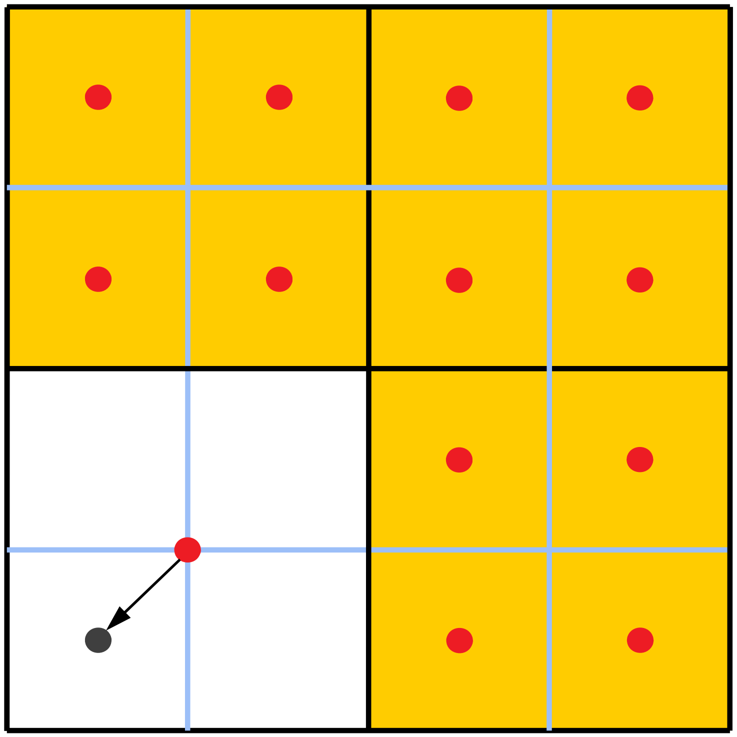
M2M

Downward pass



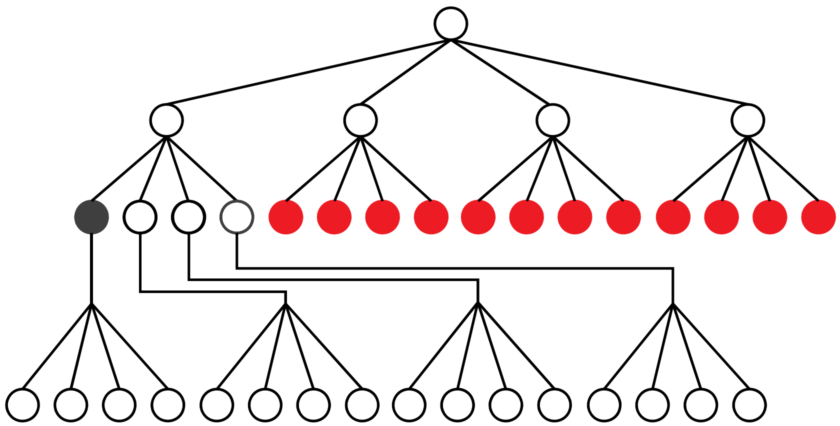
For each box, generate local expansions of the sources from its far field

- From parent (L2L)
- From interaction list (M2L)



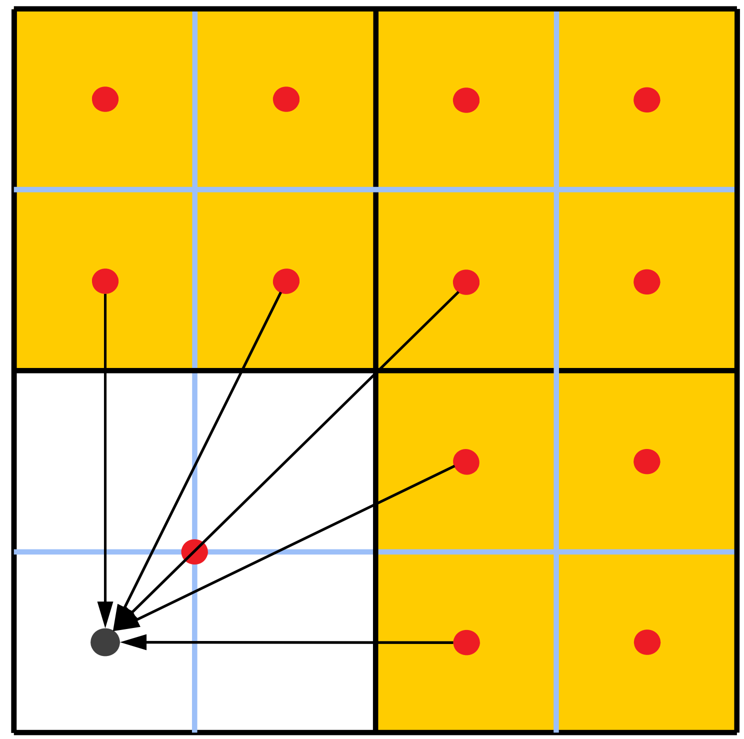
L2L

Downward pass



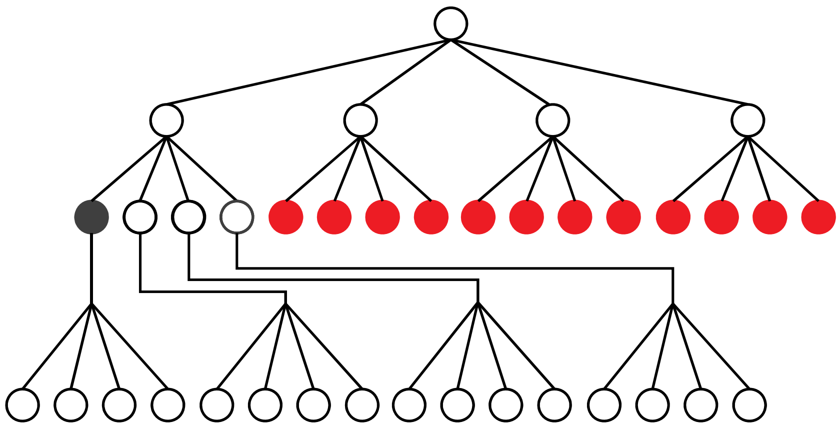
For each box, generate local expansions of the sources from its far field

- From parent (L2L)
- From interaction list (M2L)



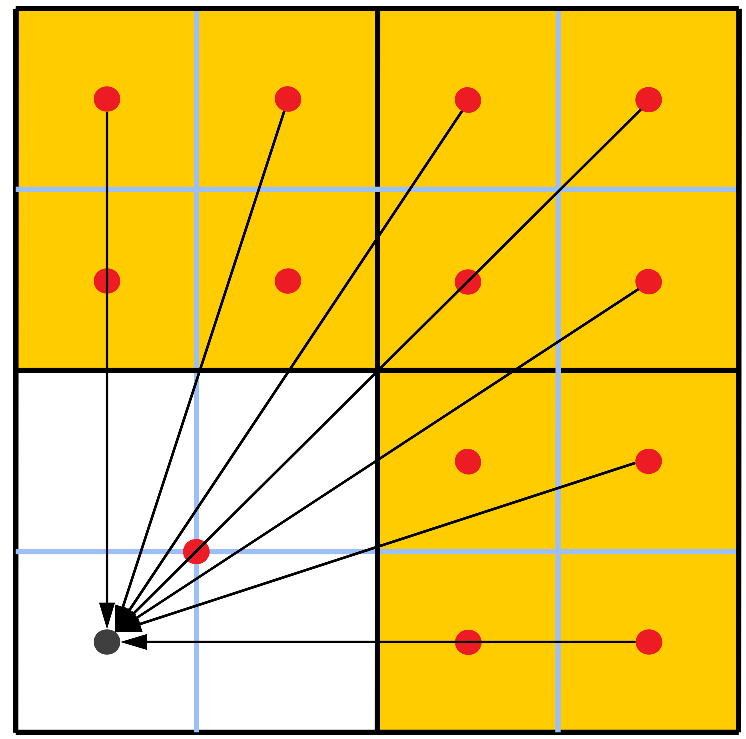
M2L

Downward pass



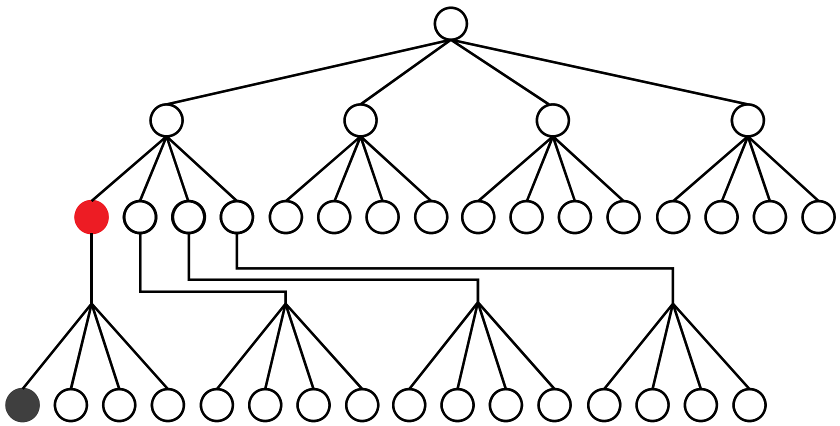
For each box, generate local expansions of the sources from its far field

- From parent (L2L)
- From interaction list (M2L)



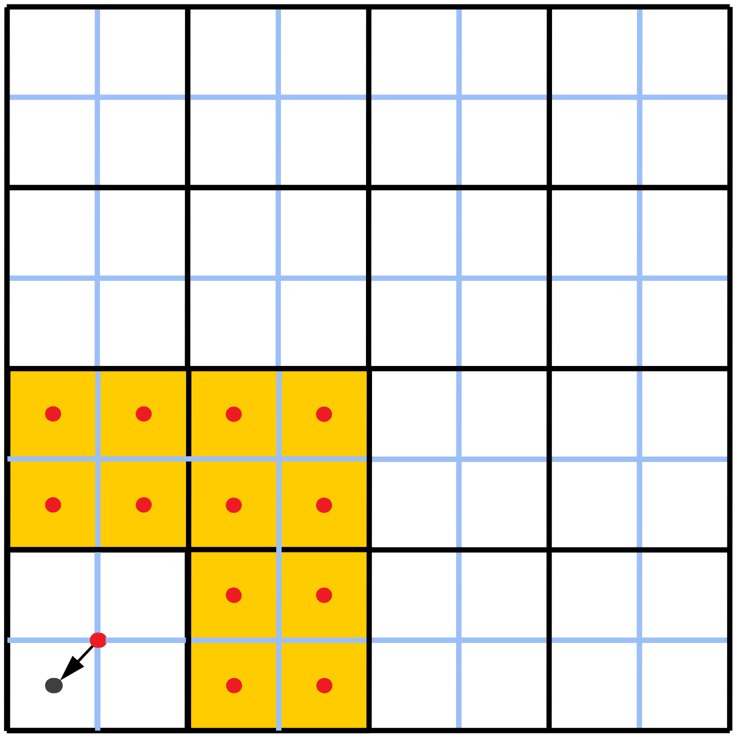
M2L

Downward pass



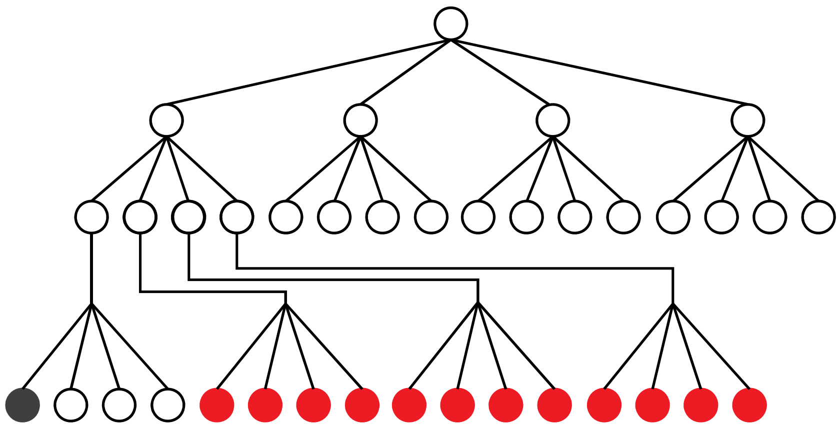
For each box, generate local expansions of the sources from its far field

- From parent (L2L)
- From interaction list (M2L)



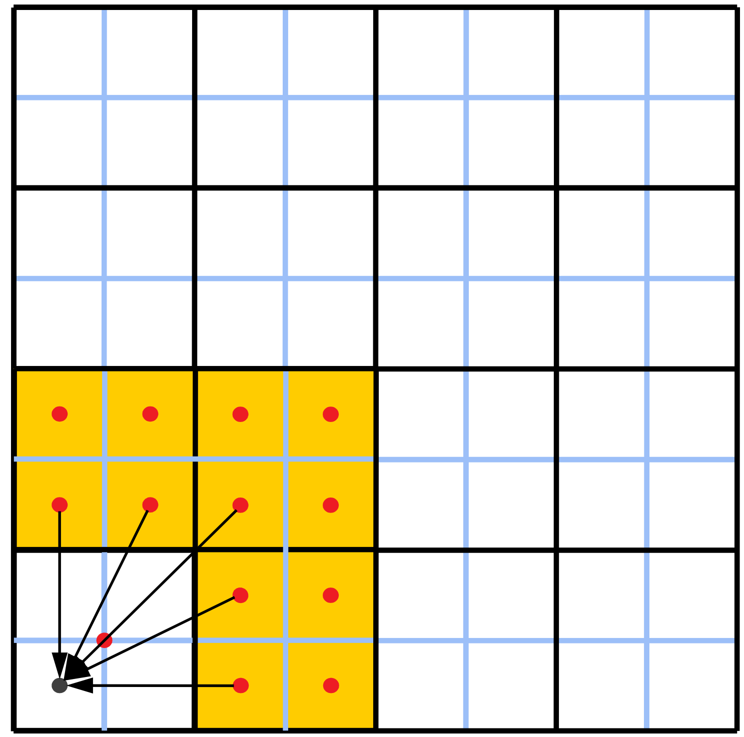
L2L

Downward pass



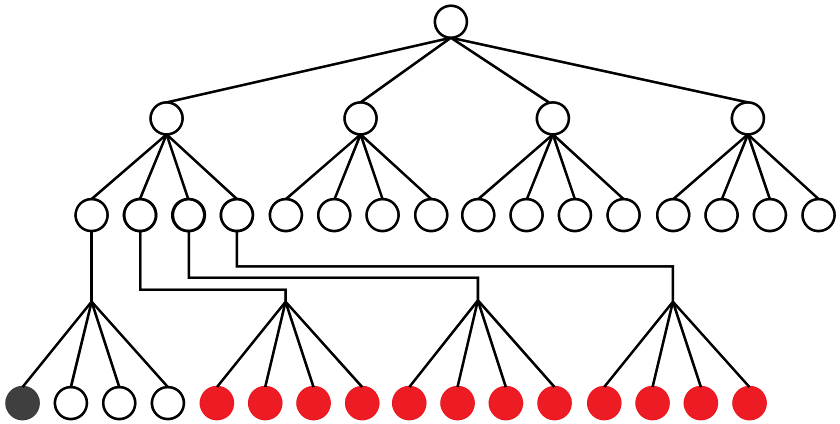
For each box, generate local expansions of the sources from its far field

- From parent (L2L)
- From interaction list (M2L)



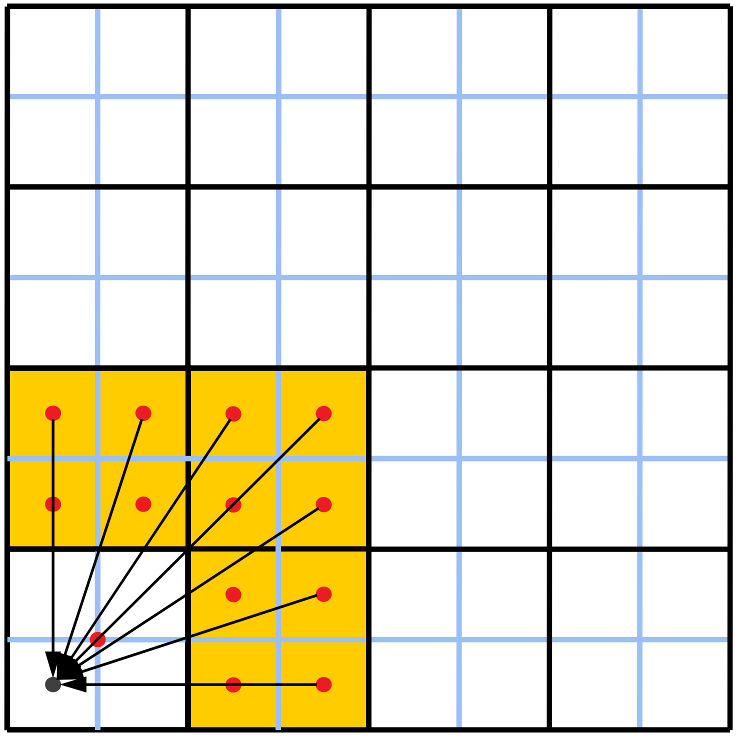
M2L

Downward pass



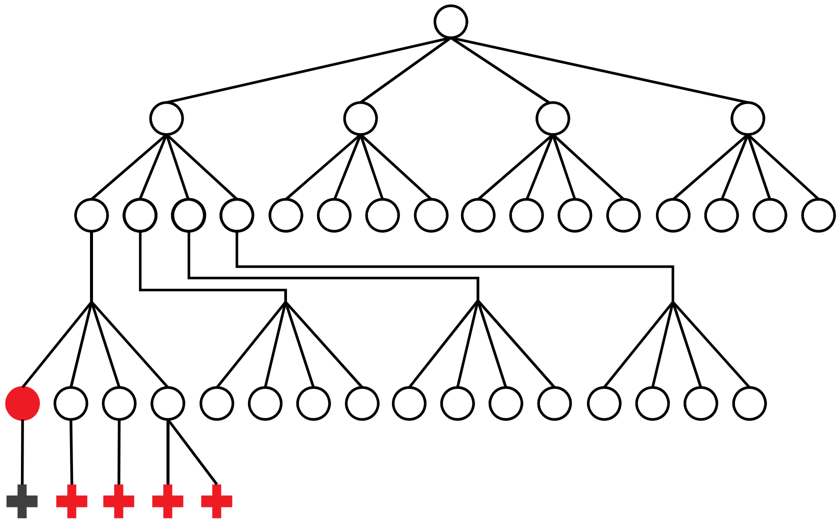
For each box, generate local expansions of the sources from its far field

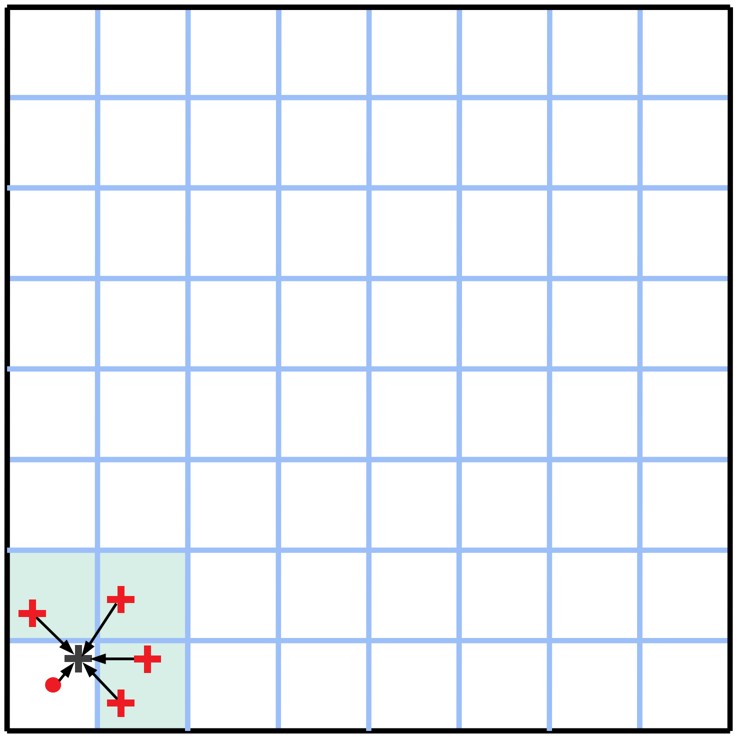
- From parent (L2L)
- From interaction list (M2L)



M2L

Downward pass





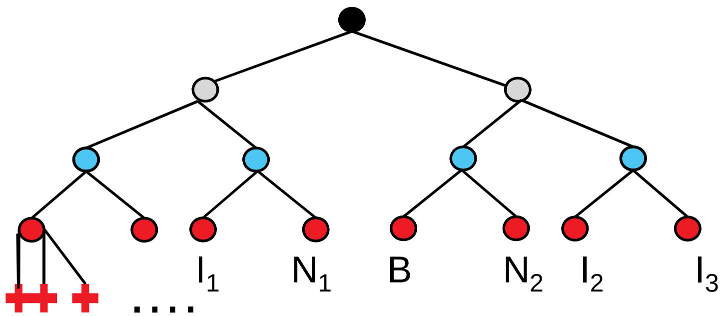
For leaf box,
evaluate potential

- From local expansion
- Contribution from neighbor list (direct evaluation)

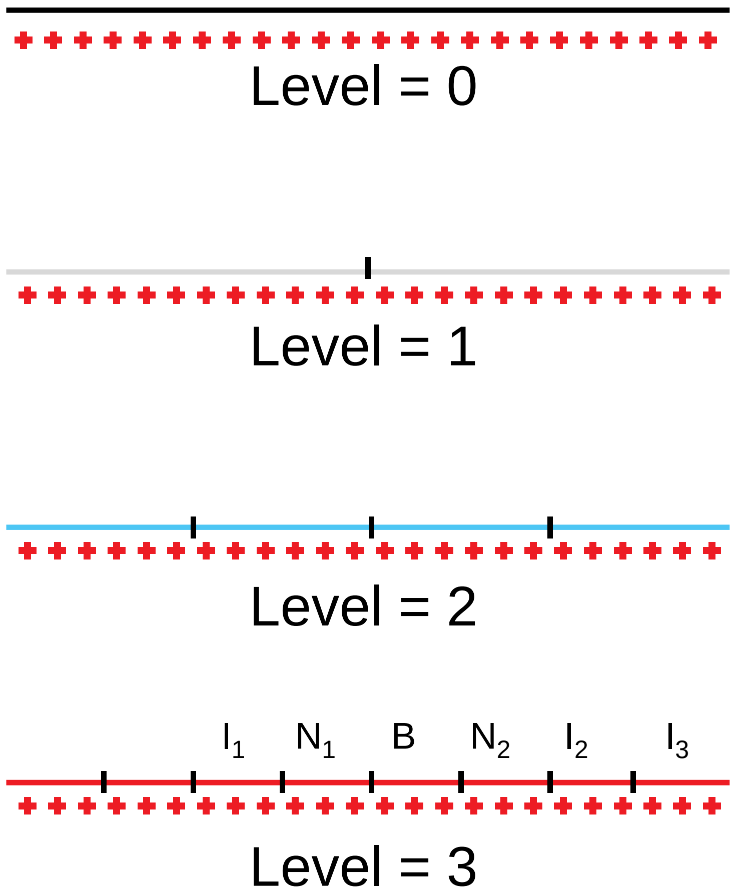
1D Version for FMM-FFT

Transform 1D domain of uniform points into binary tree fixed depth tree

- N_{leaf} points per leaf
- Build NF and FF



- The upward and downward passes trade accuracy for reduced computation and communication



Reimplementation

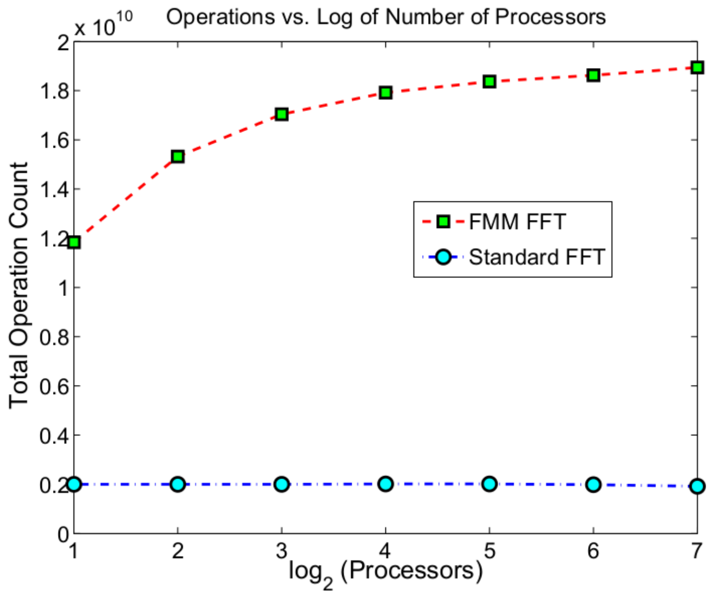
Above approach for the FMM-accelerated one dimensional FFT (FMM-FFT) has been reimplemented:

- Fortran code replaced with C;
- Local FFTs replaced with FFTW;
- Minor initial accelerations of the 1D FMM (1/cot(r/2) kernel);
- Various tests run on Intel(R) Xeon(R) CPU X5650 @ 2.67GHz CPUs and QDR InfiniBand (8 cores per node and 24GB of memory per node);
- Following are 3 tests to measure effect of varying processors, how the FMM operation load is balanced and its effect and the effect of varying FMM precision.

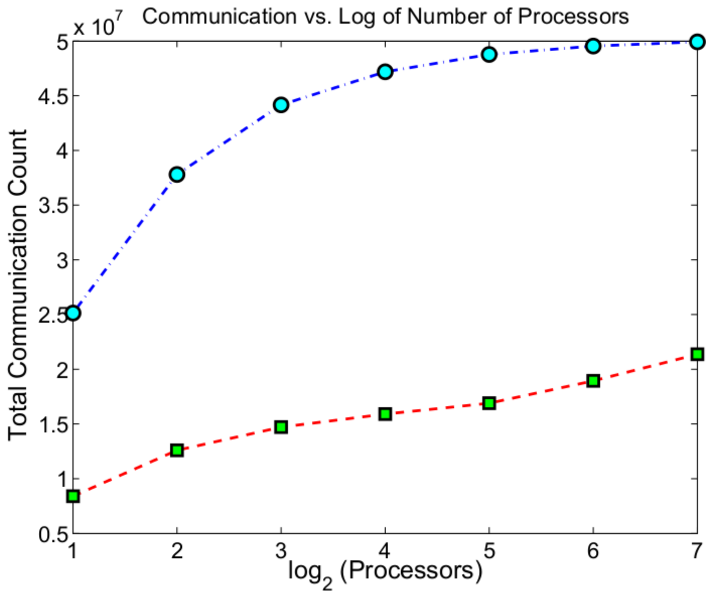
FMM-FFT Results: Fixed Problem Size, Varying Processors

Intel(R) Xeon(R) CPU X5650 @ 2.67GHz CPUs and QDR InfiniBand. 1.68 x 10^7 points, 12 digits of FMM precision and 32 points per leaf interval:

- Operation Count:



- Communication Count:

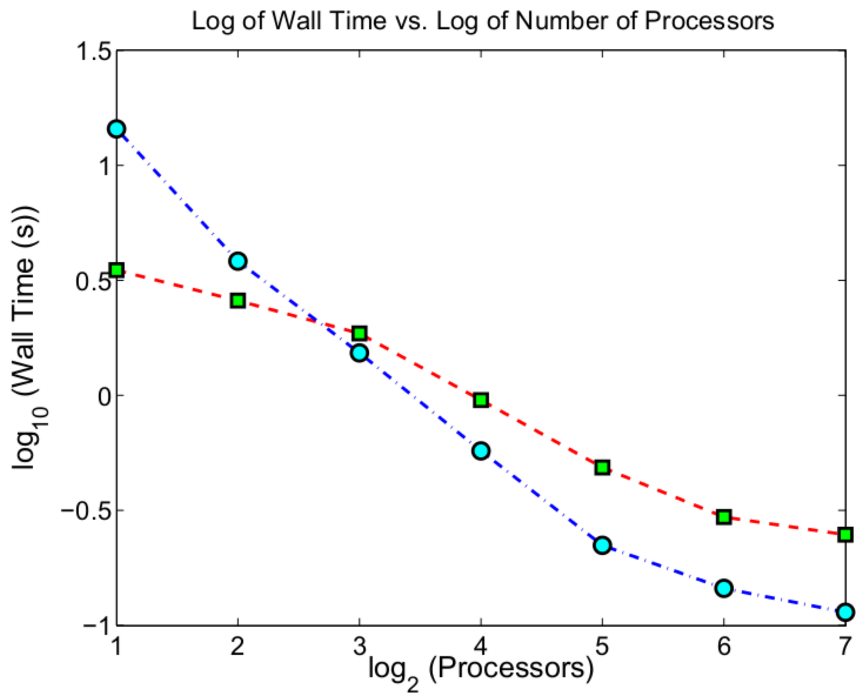


- ~10x increase in arithmetic and ~3x reduction in communication.

FMM-FFT Results: Fixed Problem Size, Varying Processors

Intel(R) Xeon(R) CPU X5650 @ 2.67GHz CPUs and QDR InfiniBand. 1.68×10^7 points, 12 digits of FMM precision and 32 points per leaf interval:

- Wall Time:



- Despite ~10x increase in arithmetic, ~3x reduction in communication, only ~2x slower runtimes.

FMM-FFT Results: Fixed Problem Size, Varying Processors

Intel(R) Xeon(R) CPU X5650 @ 2.67GHz CPUs and QDR InfiniBand. 1.68×10^7 points, 12 digits of FMM precision and 32 points per leaf interval:

- Investigate the total operation counts in the two major stage of the FMM (Near-Field and Far-Field):

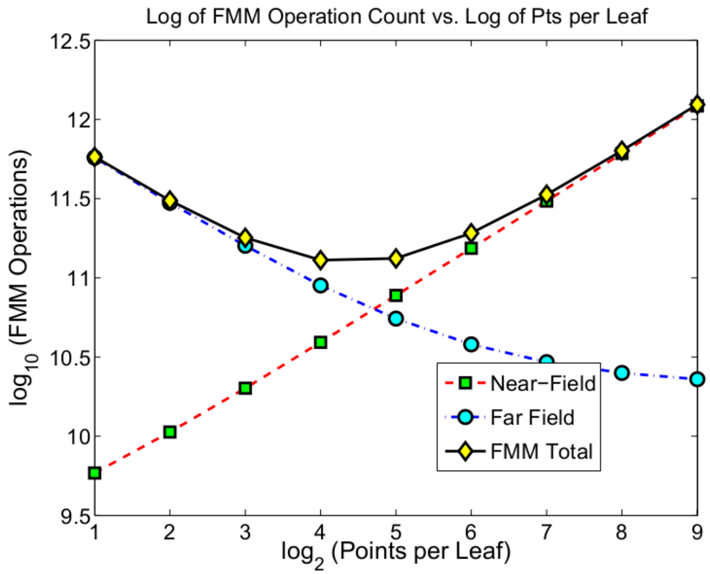
NP	NF^{ops}	FF^{ops}
2	2.53×10^9	2.39×10^9
4	1.86×10^9	1.47×10^9
16	5.75×10^8	4.19×10^8
32	2.97×10^8	2.14×10^8
64	1.51×10^8	1.09×10^8
128	7.60×10^7	5.66×10^7

- These operation loads can be skewed as desired!

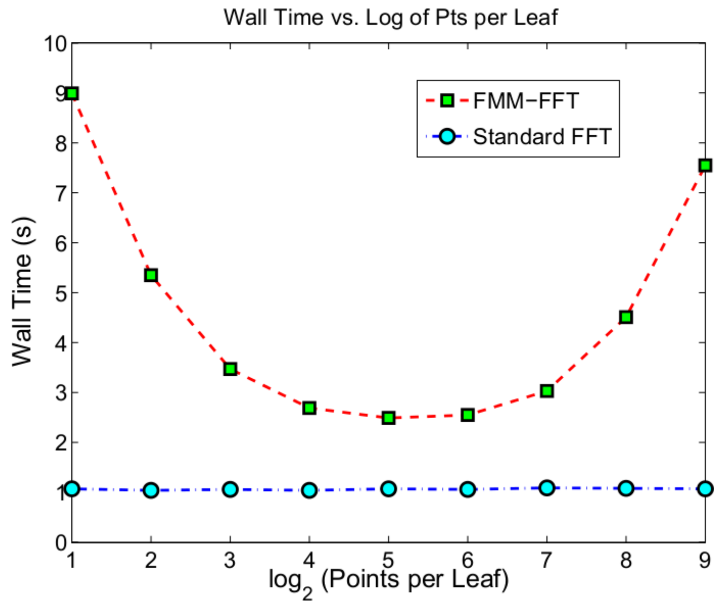
FMM-FFT Results: Fixed Problem Size, Varying FMM Near-Field and Far-Field Loads

Intel(R) Xeon(R) CPU X5650 @ 2.67GHz CPUs and QDR InfiniBand. 1.34 x 10^8 points, 12 digits of FMM precision and 64 processors:

- Operation Count:



- Wall Time:

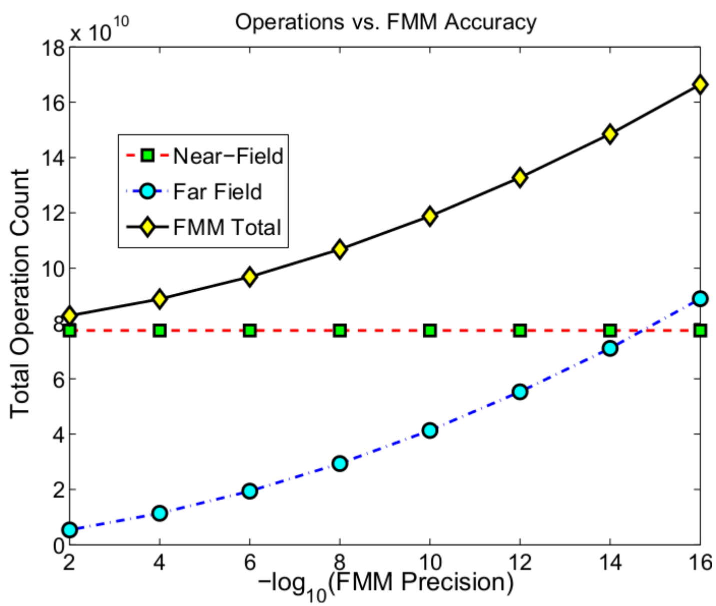


- The optimal operation count when the NF and FF processes occur in order is optimized by the leaf param.

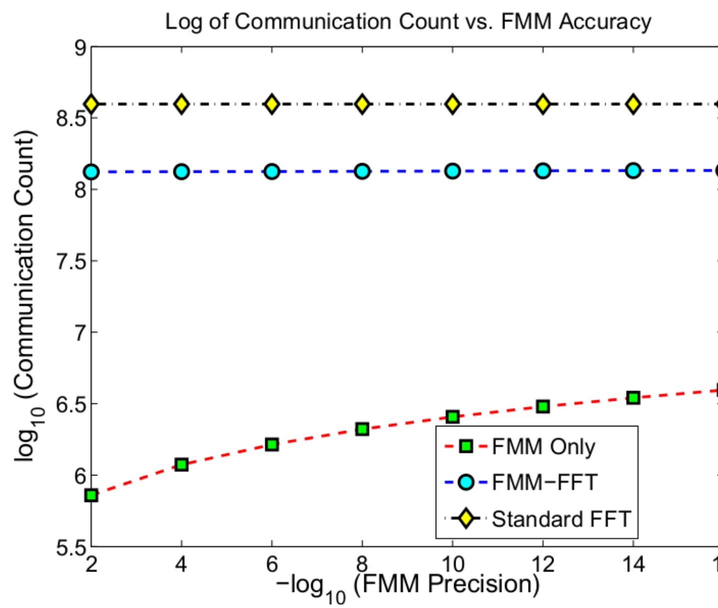
FMM-FFT Additional Results: Fixed Problem Size and Processors, Varying FMM Accuracy

Intel(R) Xeon(R) CPU X5650 @ 2.67GHz CPUs and QDR InfiniBand. 1.34 x 10^8 points, 12 digits of FMM precision, 64 processors and 32 points per leaf interval:

- Operation Count:



- Communication Count:

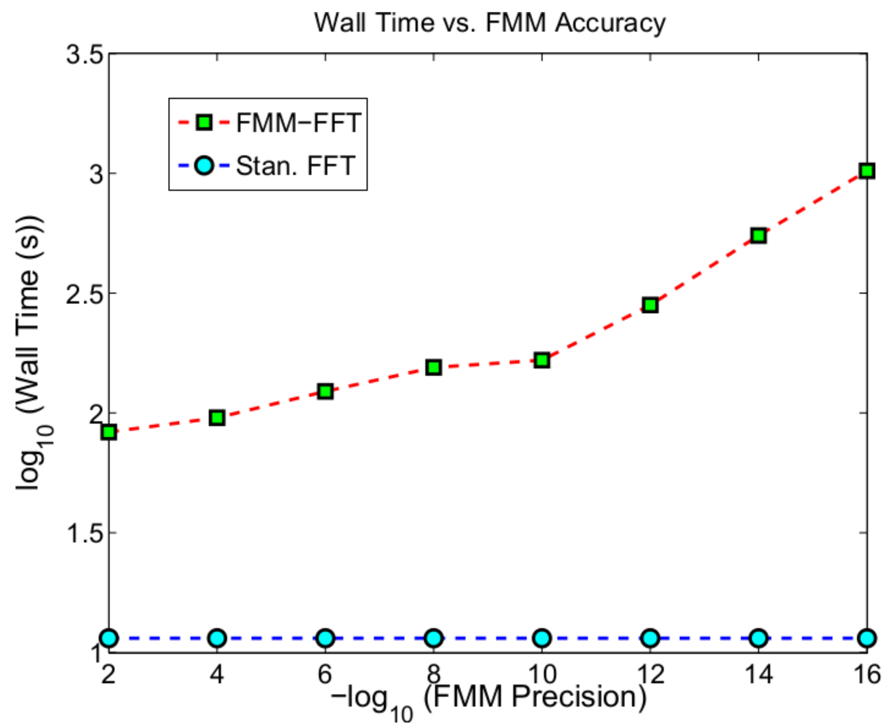


- Communication effect is minimal with greater FMM accur.

FMM-FFT Additional Results: Fixed Problem Size and Processors, Varying FMM Accuracy

Intel(R) Xeon(R) CPU X5650 @ 2.67GHz CPUs and QDR InfiniBand. 1.34×10^8 points, 12 digits of FMM precision, 64 processors and 32 points per leaf interval:

- Wall Time:

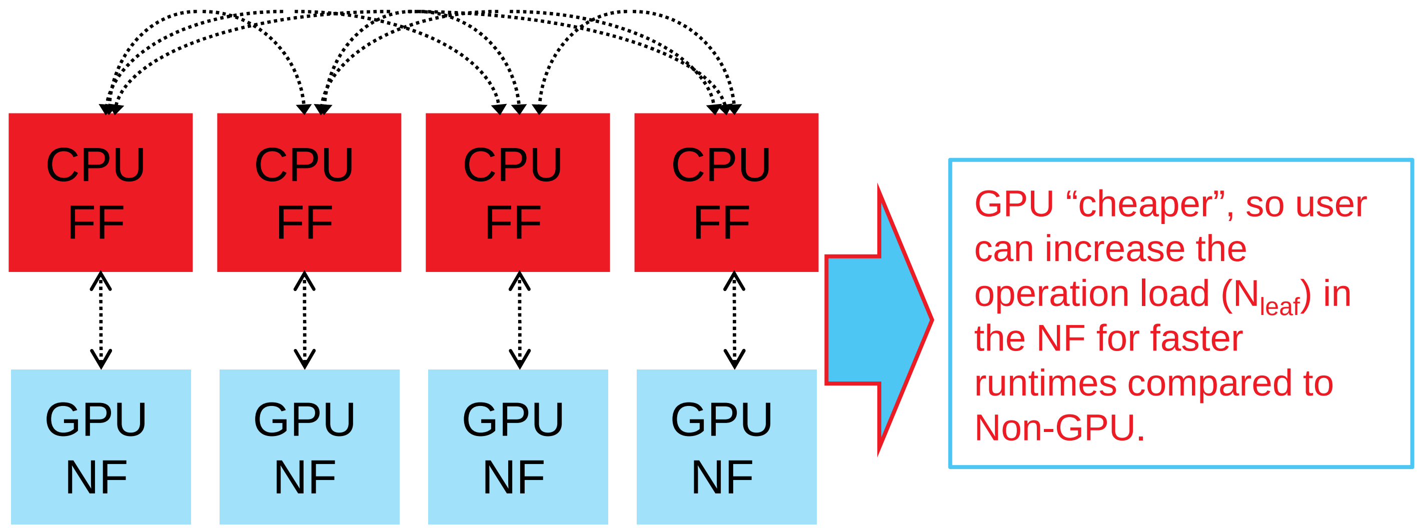


- Wall time effect changes rapidly with higher precision.

Refactoring FMM-FFT for GPU

Again, FMM Has Two Distinct Steps

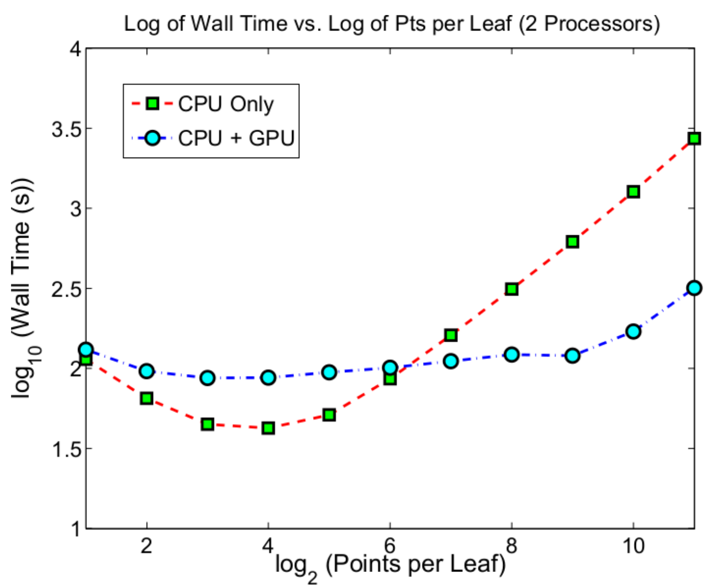
- Near-Field (NF) and Far-Field (FF) can be potentially computed separately if resources available and synched upon completion.
- Refactoring Approach: FF operations rely on MPI and remain on CPU. NF operations are pushed to GPU.
- Basic Idea as inspired by Lashuk et. al. (SC’09):



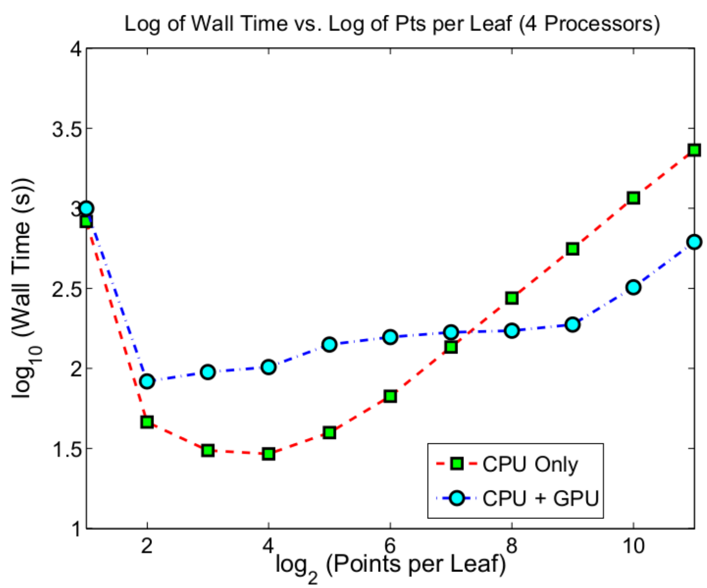
GPU FMM-FFT Results: Fixed Problem Size, Varying FMM Near-Field and Far-Field Loads.

Up to 4 Intel(R) Xeon(R) CPU X5650 @ 2.67GHz CPUs, each with a dedicated 448 thread Tesla-M2070 GPU node and QDR InfiniBand. 1.34×10^8 points, 12 digits of FMM precision:

- 2 processors:



- 4 processors:



- For large NF loads, GPU FMM-FFT nearly ~10x faster.

Conclusions

Current and Future Work

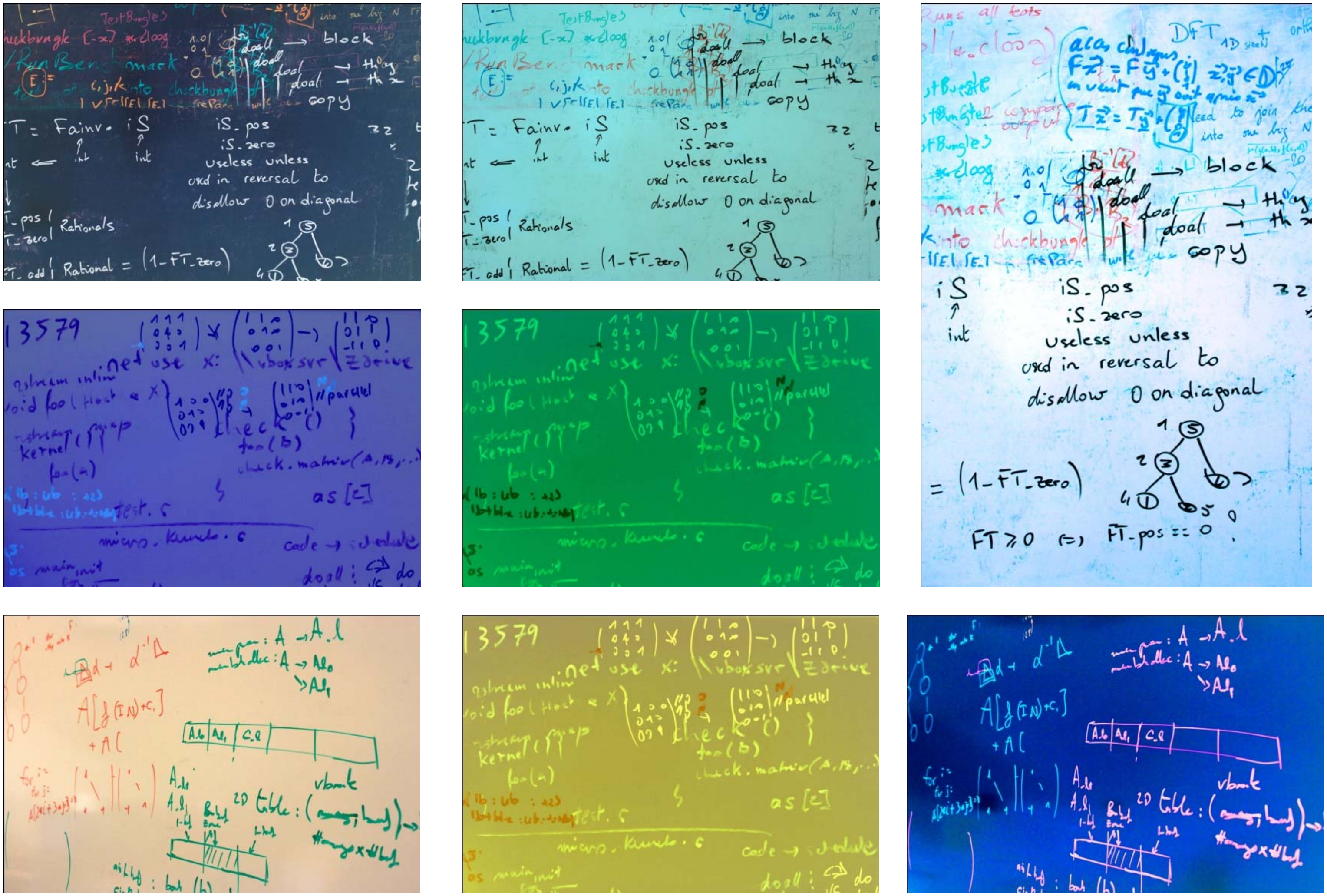
- Showed results for reimplementation of FMM-FFT from Edelman et. al. as well investigated how loads are balanced between the NF and FF for the FMM.
- Showed how NF and FF contributions can be asynchronously computed on the GPU and CPU along with initial promising results for a small number of processors.
- Ongoing:
  - GPU optimizations and R-Stream (Meister et. al., 2011) incorporations;
  - FMM optimizations (including symmetry exploitations);
  - Further tests with more processors;
  - Comparisons to other approaches;
  - Collaborations with other groups, including in expanding to higher dimensions.

Acknowledgements

The following were instrumental in this work:

- Alan Edelman in sharing the original Fortran code;
- Leslie Greengard and the Fast Algorithms group at New York University’s Courant Institute;
- Peter Tang at Intel for mentioning this work in their SC’12 paper and encouraging pursuing its reimplementation;
- Reservoir’s R-Stream team;
- Sponsored by Defense Advanced Research Projects Agency, Microsystems Technology Office (MTO). Program: Power Efficiency Revolution for Embedded Computing Technologies (PERFECT). Issued by DARPA/CMO under Contract No: HR0011-12-C-0123.
- Great reviewers!

Reservoir Labs

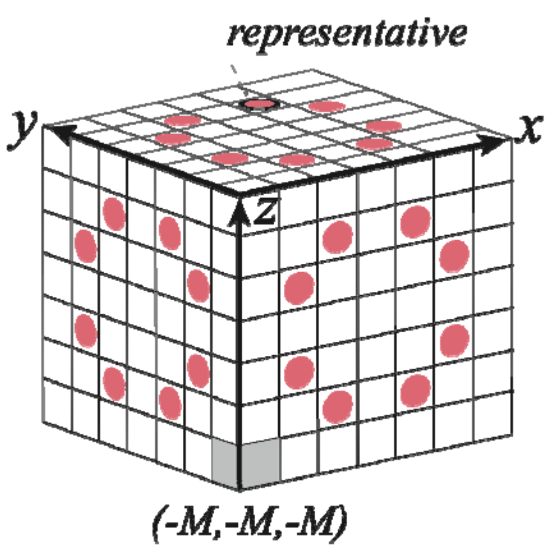


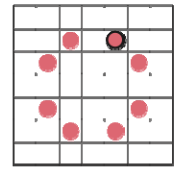
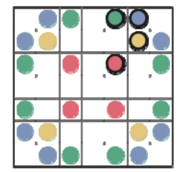
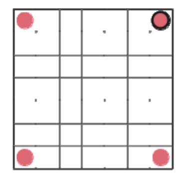
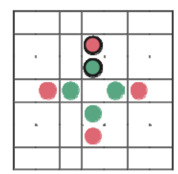
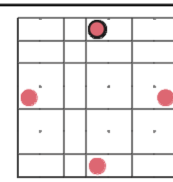
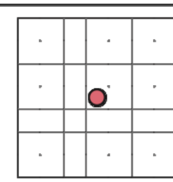
Extra Slides

Symmetries

Higher dimensional example:

- Up to rotation and reflection, many pairs of interactions are equivalent



Series signatures	7^3 layer classes	Reference cube
(i, j, k)		$(\lvert i\rvert, \lvert j\rvert, \lvert k\rvert)$, $\lvert i\rvert < \lvert j\rvert < \lvert k\rvert$
(i, i, j)		$(\lvert i\rvert, \lvert i\rvert, \lvert j\rvert)$,$\lvert i\rvert < \lvert j\rvert$ or $(\lvert i\rvert, \lvert j\rvert, \lvert j\rvert)$
(i, i, i)		$(\lvert i\rvert, \lvert i\rvert, \lvert i\rvert)$
$(0, i, j)$		$(0, \lvert i\rvert, \lvert j\rvert)$
$(0, i, i)$		$(0, \lvert i\rvert, \lvert i\rvert)$
$(0, i, i)$		$(0, 0, \lvert i\rvert)$
$(0, 0, 0)$	—	$(0, 0, 0)$