GPUs in Computational Science

Matthew Knepley[1] and Felipe Cruz[2]

[1]Computation Institute
University of Chicago

[2]Nagasaki Advnaced Computing Center
Nagasaki University

International Workshop on GPU Solutions to
Multiscale Problems in Science and Engineering
Harbin, China, July 27, 2010

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
- Parallel FMM
  - Provably scalable formulation
  - Complete reuse of serial code

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 25% 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:

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

- 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

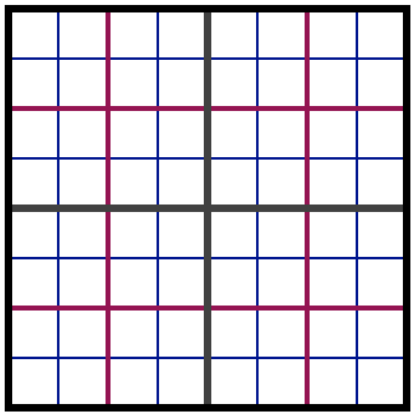
PetFMM

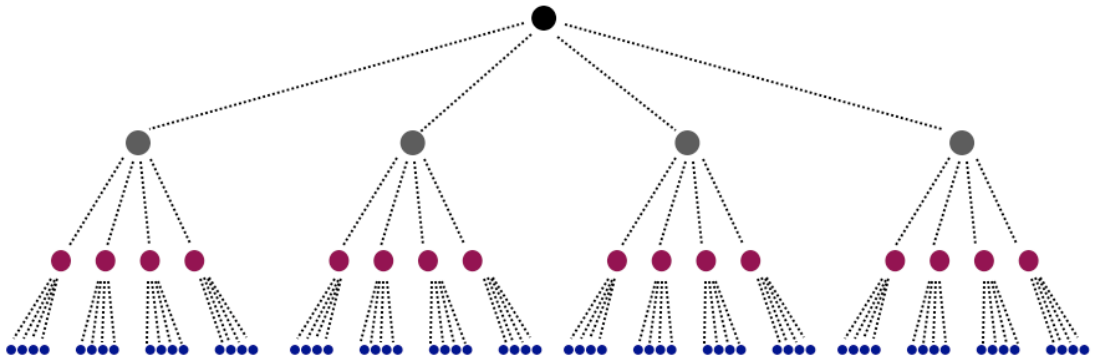
PetFMM is an freely available implementation of the
Fast Multipole Method
http://barbagroup.bu.edu/Barba_group/PetFMM.html

- 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

Spatial Decomposition

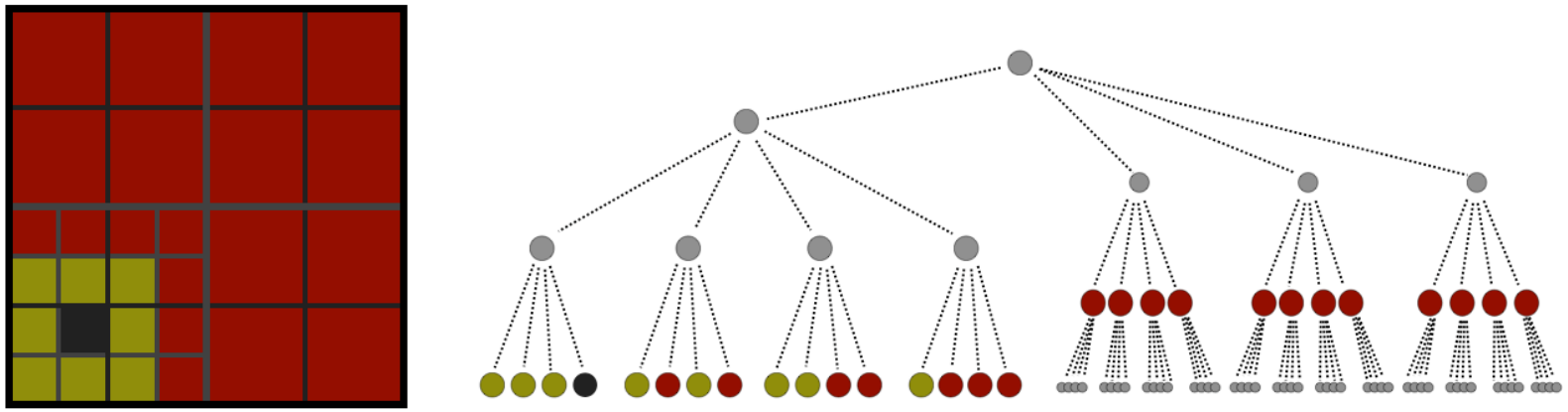
Pairs of boxes are divided into *near* and *far*:





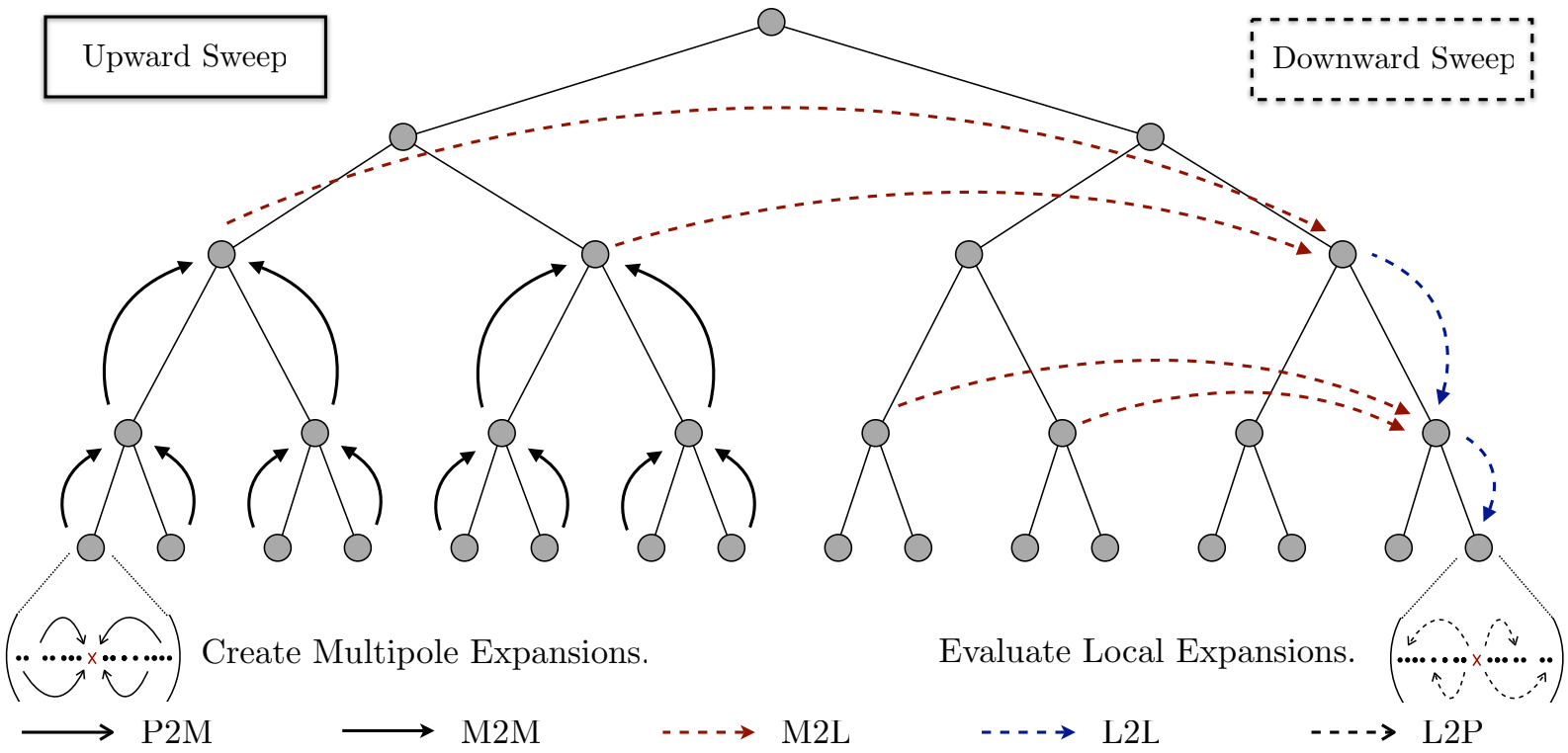
Spatial Decomposition

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



Neighbors are treated as *very near*.

Functional Decomposition

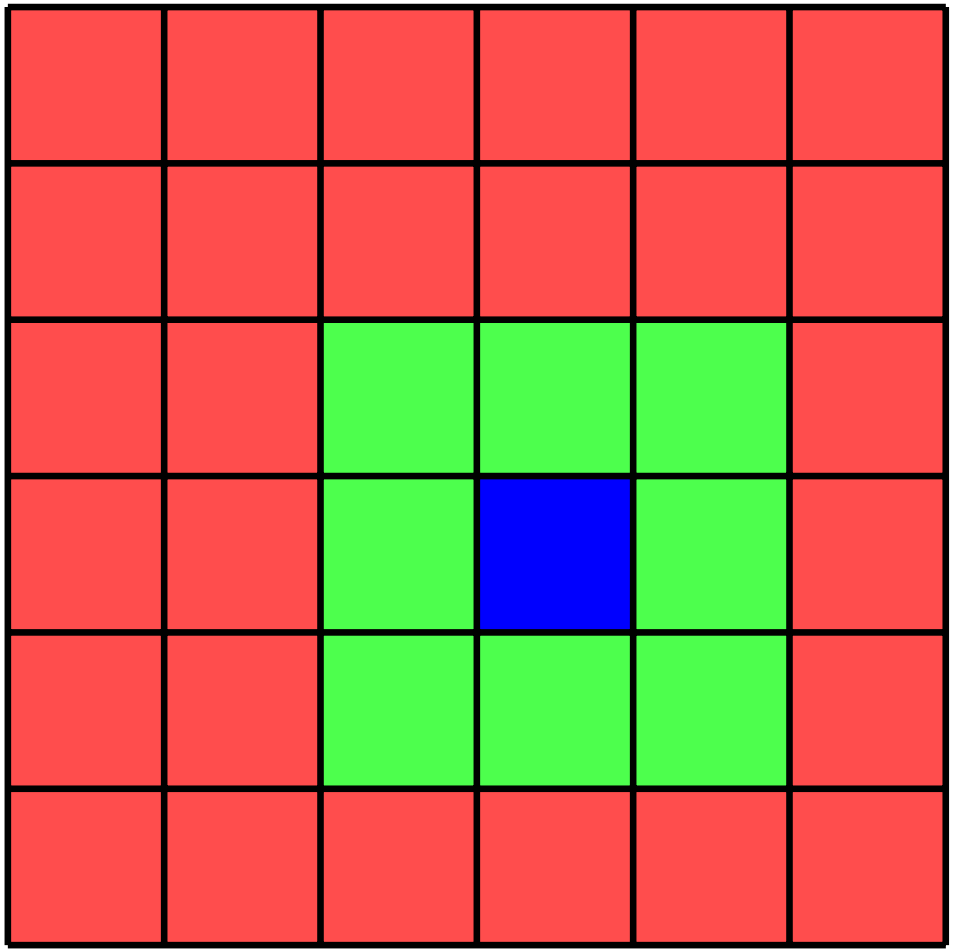


Outline

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

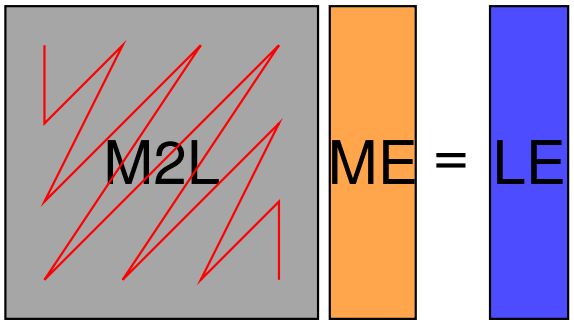
Memory limits concurrency!

GPU M2L
Version 1

Apply M2L transform matrix-free

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

- 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

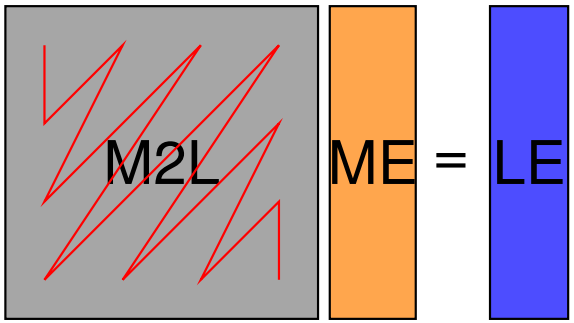


GPU M2L
Version 1

Apply M2L transform matrix-free

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

- 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

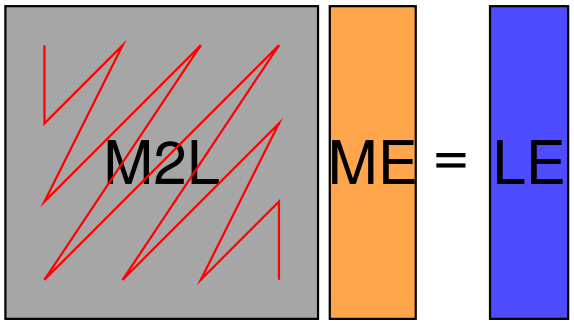


GPU M2L
Version 1

Apply M2L transform matrix-free

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

- 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

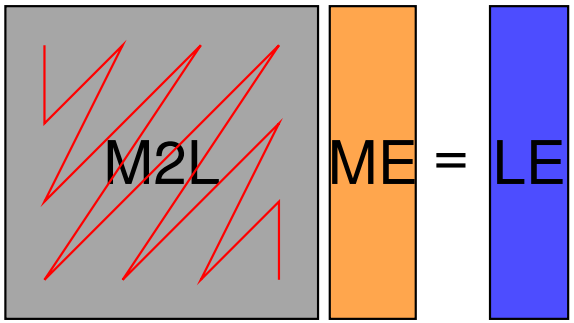


GPU M2L
Version 1

Apply M2L transform matrix-free

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

- 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

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

- 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

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

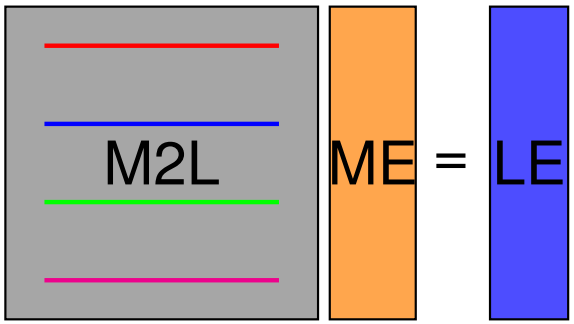
- Move 27 LE to global memory per TB
- $27 \times 2p = 648$ floats
- With 32 threads, takes 21 memory transactions

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

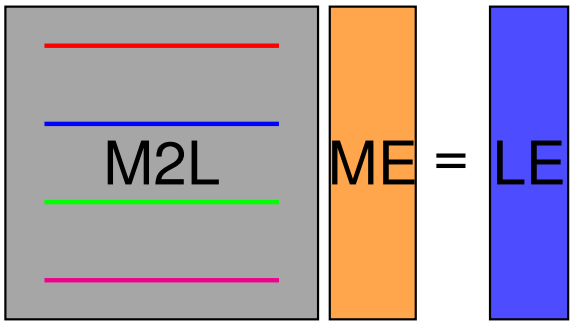


GPU M2L
Version 2

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

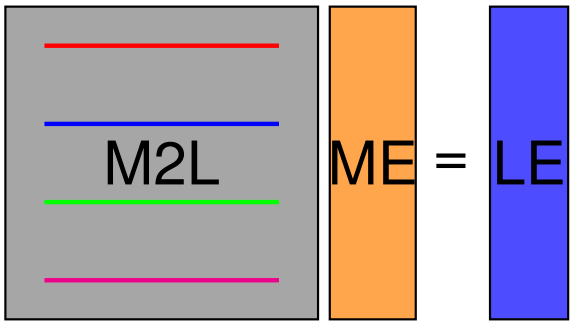


GPU M2L
Version 2

One thread per *element* of the LE

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

- 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

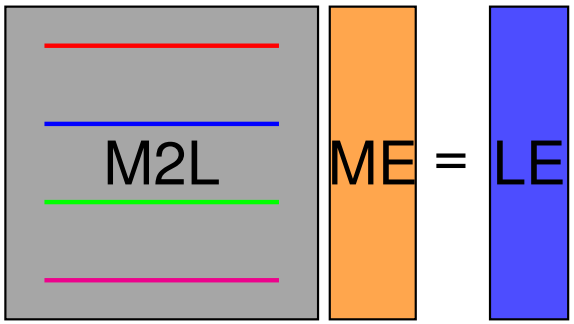


GPU M2L
Version 2

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

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

- 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

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