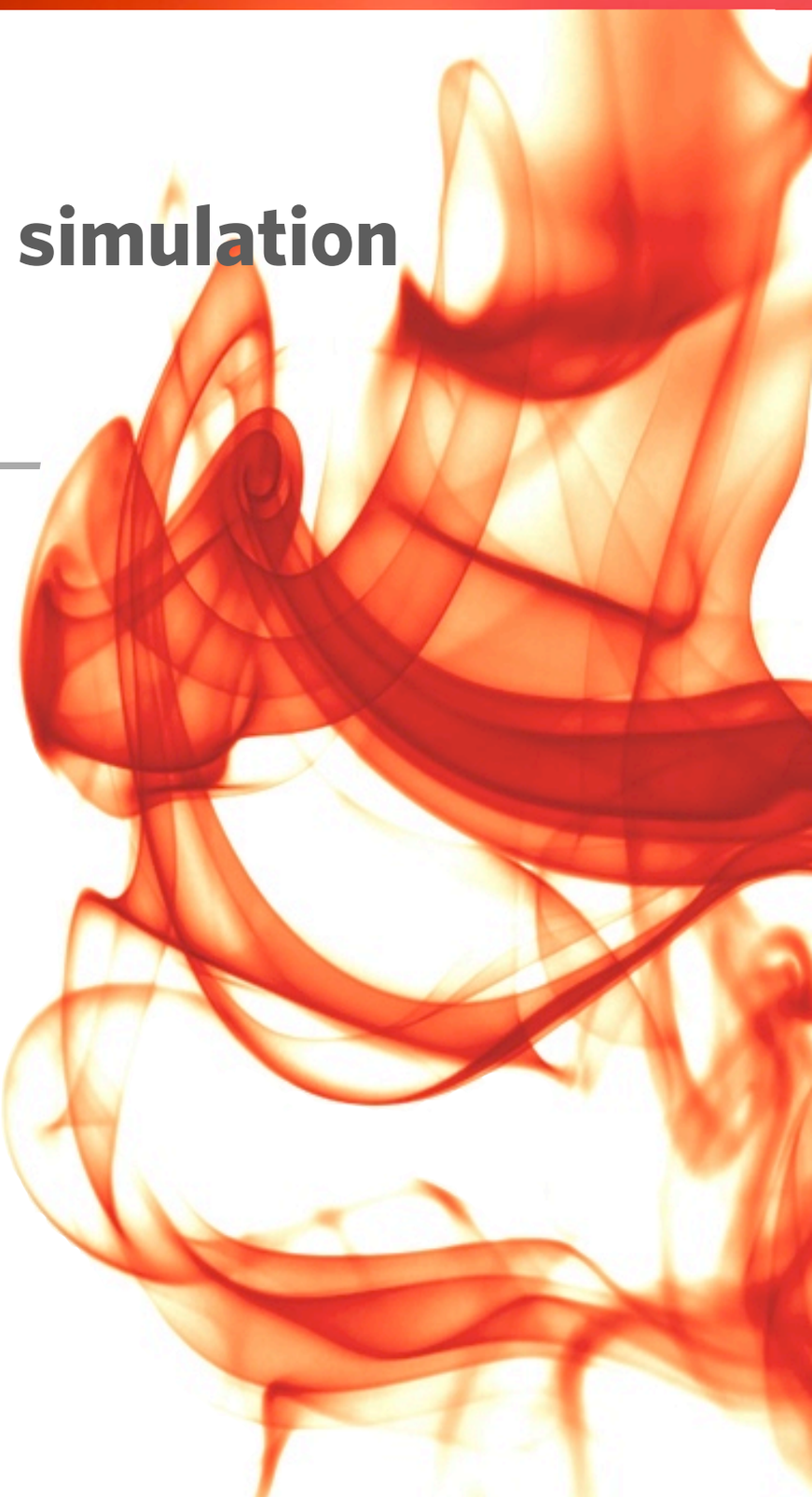
Center for Computational Science

BOSTON UNIVERSITY

Toward GPU-accelerated meshfree fluids simulation using the fast multipole method



Lorena A Barba

Boston University Department of Mechanical Engineering

with:

Felipe Cruz, PhD student
Simon Layton, PhD student
Rio Yokota, postdoc

BU College of Engineering

Topics

- Meshfree method for fluids simulation — vortex method
- Computational challenge
  - N-body summation involved
- Innovation using new hardware
  - scientific computing on graphics cards (GPU)

Meshfree method for fluid simulation

BU College of Engineering

Vortex method for fluid simulation

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = -\frac{\nabla p}{\rho} + \nu \nabla^2 u$$

Vortex method for fluid simulation

- particle method for incompressible, Newtonian fluid

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = -\frac{\nabla p}{\rho} + \nu \nabla^2 u$$

Vortex method for fluid simulation

- particle method for incompressible, Newtonian fluid

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = -\frac{\nabla p}{\rho} + \nu \nabla^2 u$$

- based on vorticity, $\omega = \nabla \times u$

Vortex method for fluid simulation

- particle method for incompressible, Newtonian fluid

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = -\frac{\nabla p}{\rho} + \nu \nabla^2 u$$

- based on vorticity, $\omega = \nabla \times u$

$$\frac{\partial \omega}{\partial t} + u \cdot \nabla \omega = \omega \cdot \nabla u + \nu \nabla^2 \omega$$

- Vorticity transport equation

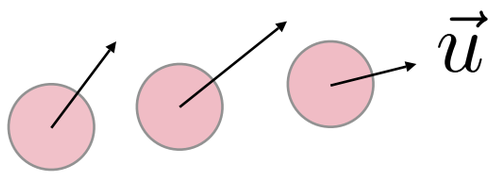
$$\frac{\partial \omega}{\partial t} + u \cdot \nabla \omega = \omega \cdot \nabla u + \nu \nabla^2 \omega$$

- 2D ideal case ➛ $$\frac{\mathrm{D}\omega}{\mathrm{D}t} = 0$$

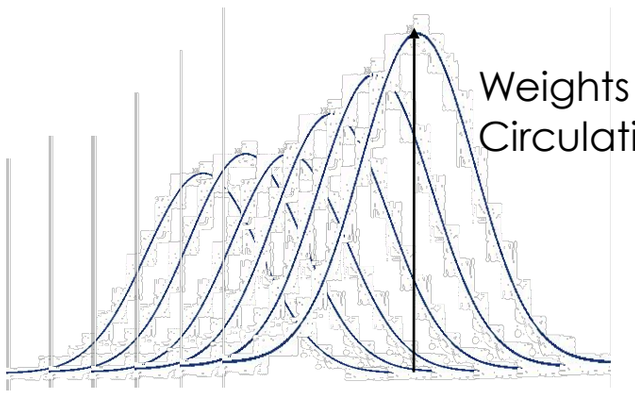
  - if velocity is known for a fluid element at x_i
  - vorticity transport automatically satisfied by $$\frac{dx_i}{dt} = u(x_i, t)$$

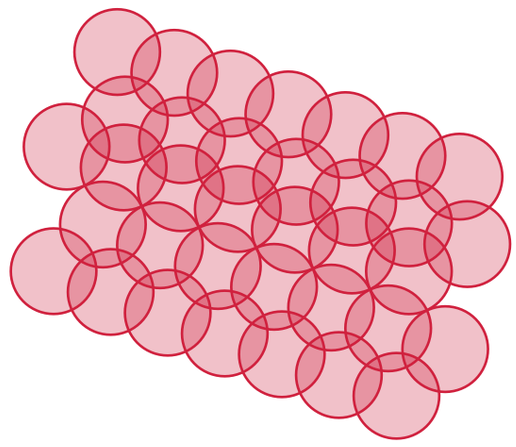
- Vortex method discretization
  - express vorticity as → $\omega_\sigma(x,t) = \sum_{i=1}^{N} \gamma_i \zeta_\sigma(x - x_i)$
  - interpreted as "particles" ω

$$\frac{dx_i}{dt} = u(x_i, t)$$



  - Gaussian distribution $\zeta_\sigma(x) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{|x|^2}{2\sigma^2}\right)$





- Find velocity from vorticity: invert $\omega = -\nabla^2 \psi$

  - in 2D $$u(x) = -\frac{1}{2\pi} \int \frac{(x - x') \times \omega(x') \hat{e}_z}{|x - x'|^2} dx'$$

$$\omega_\sigma(x,t) = \sum_{i=1}^{N} \gamma_i \zeta_\sigma(x - x_i)$$

  - get:

$$u_\sigma(x,t) = \sum_{i=1}^{N} \gamma_i \mathbb{K}_\sigma(x - x_i)$$

with

$$\mathbb{K}_\sigma = \frac{1}{2\pi|x|^2}(-x_2, x_1)\left(1 - \exp\left(-\frac{|x|^2}{2\sigma^2}\right)\right)$$

- Find velocity from vorticity: invert $\omega = -\nabla^2 \psi$

  - in 2D $$u(x) = -\frac{1}{2\pi} \int \frac{(x - x') \times \omega(x') \hat{e}_z}{|x - x'|^2} dx'$$

$$\omega_\sigma(x,t) = \sum_{i=1}^{N} \gamma_i \zeta_\sigma(x - x_i)$$

  - get:

$$u_\sigma(x,t) = \sum_{i=1}^{N} \gamma_i \mathbb{K}_\sigma(x - x_i)$$

with

$$\mathbb{K}_\sigma = \frac{1}{2\pi|x|^2}(-x_2, x_1)\left(1 - \exp\left(-\frac{|x|^2}{2\sigma^2}\right)\right)$$

Challenge:
Calculating the velocity
$\Rightarrow$ N-body problem

Advantages

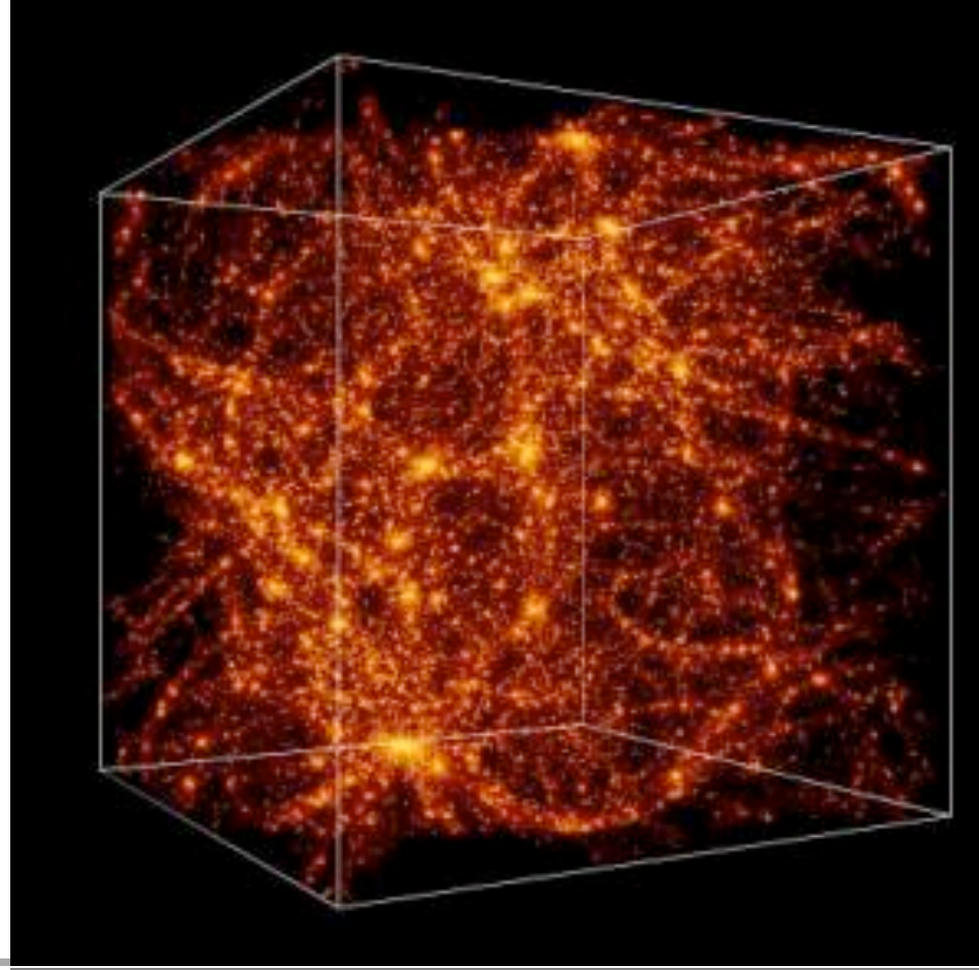
- No mesh!
- Low numerical diffusion
  - traditional CFD methods

$$\epsilon \propto \nabla^2 \omega$$

$$\frac{\partial \omega}{\partial t} + u \cdot \nabla \omega = \omega \cdot \nabla u + \nu \nabla^2 \omega$$

Consider: Helicopter rotor-tip vortices

Source — U.S. Navy's Digital Image site

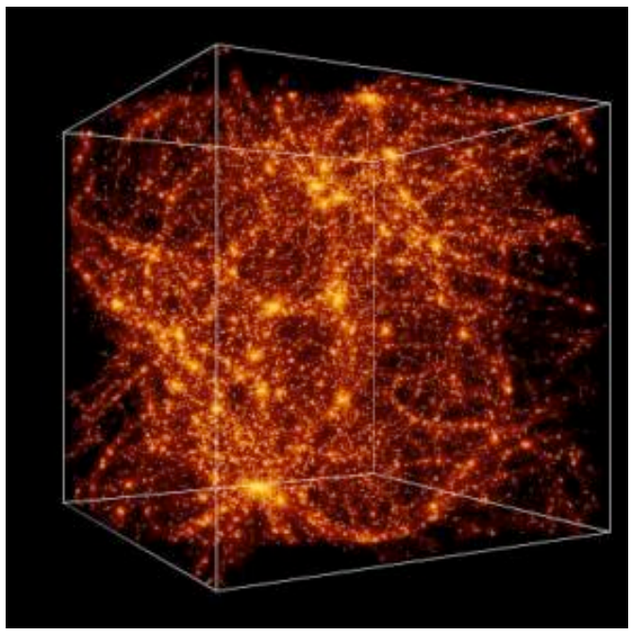


Fast solution of N-body problem

BU College of Engineering

Fast multipole method

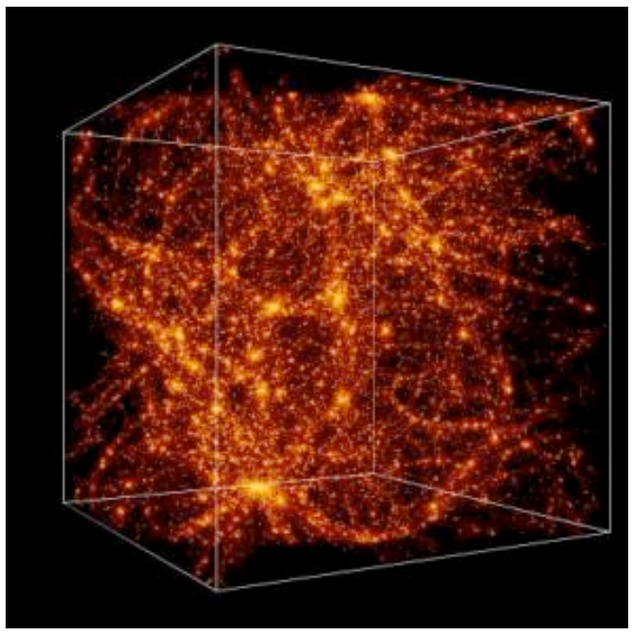
- Solves N-body problems
  - e.g. astrophysical gravity interactions
  - reduces operation count from O(N^2) to O(N)



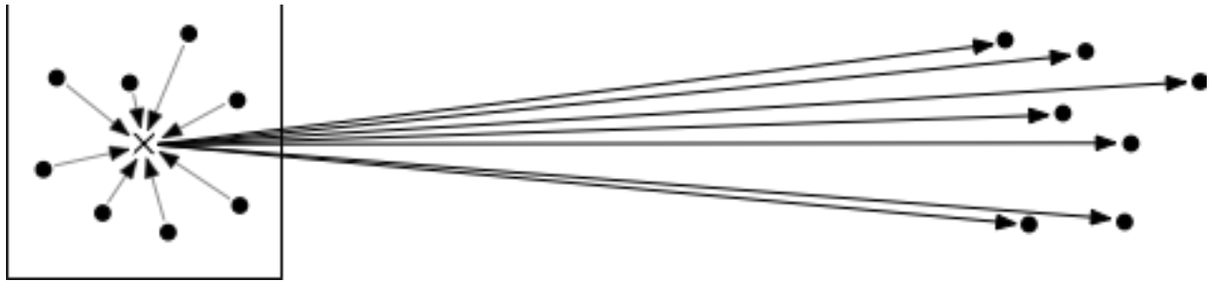
$$f(y) = \sum_{i=1}^{N} c_i \mathbf{K}(y - x_i) \qquad y \in [1...N]$$

Fast multipole method

- Solves N-body problems
  - e.g. astrophysical gravity interactions
  - reduces operation count from O(N^2) to O(N)

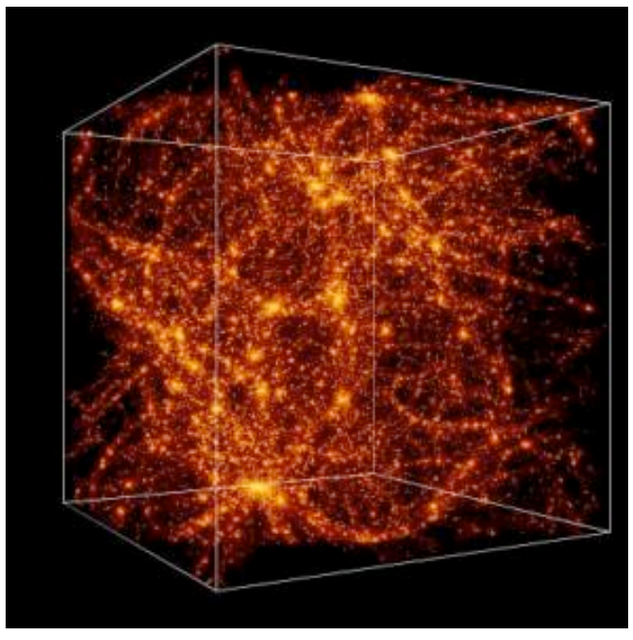


$$f(y) = \sum_{i=1}^{N} c_i \mathbf{K}(y - x_i) \qquad y \in [1...N]$$



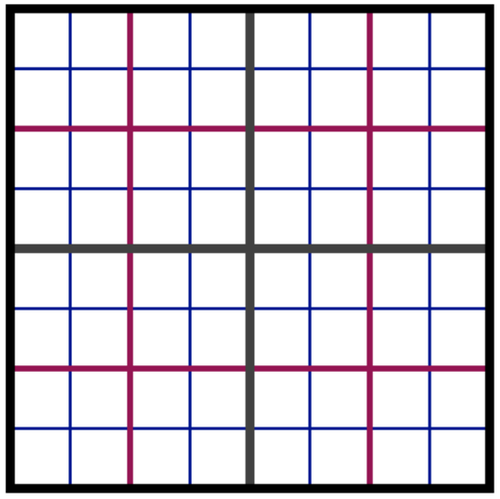
Fast multipole method

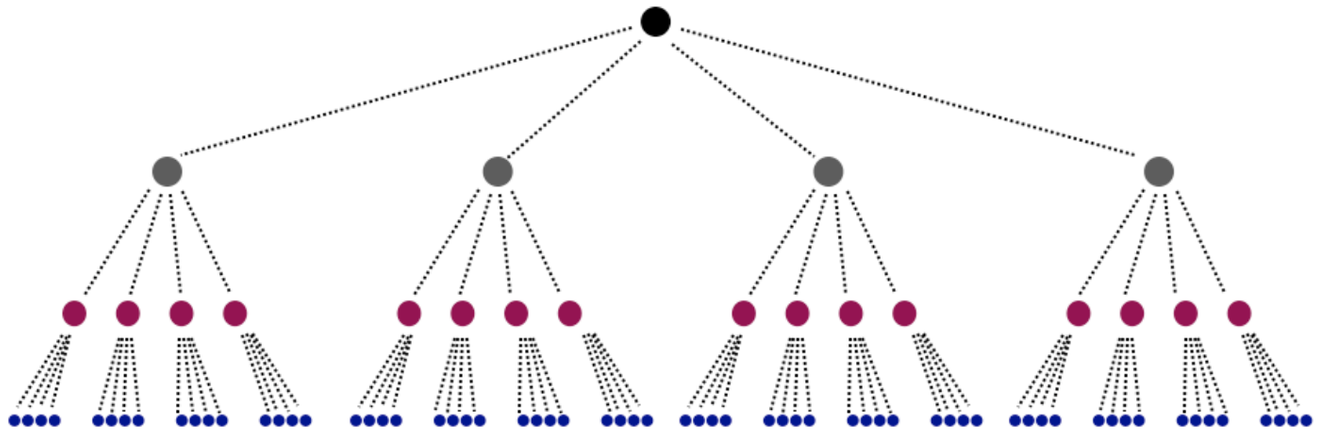
- Solves N-body problems
  - e.g. astrophysical gravity interactions
  - reduces operation count from O(N^2) to O(N)

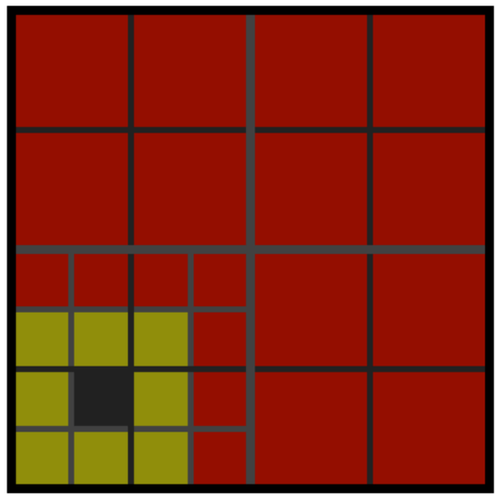


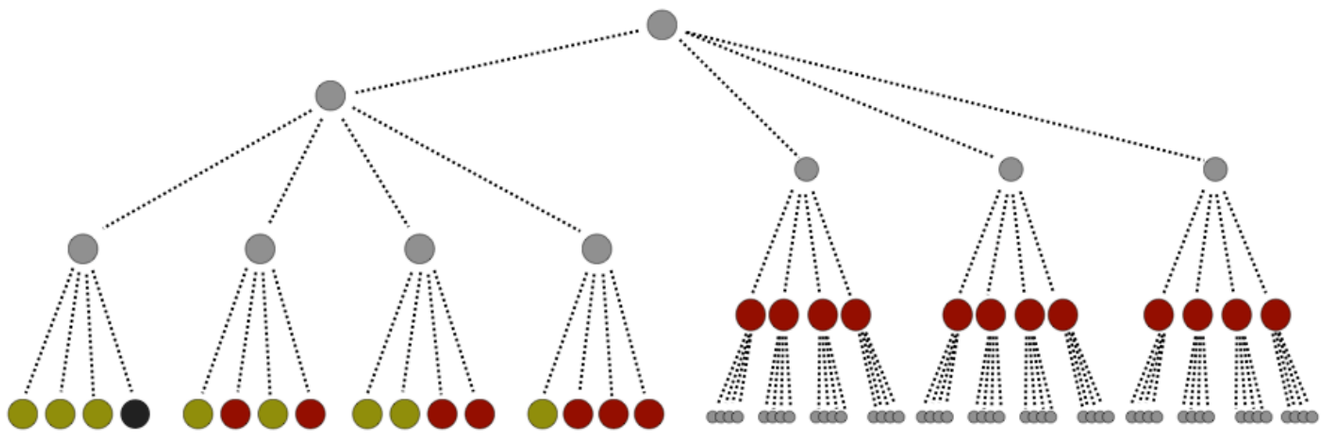
$$f(y) = \sum_{i=1}^{N} c_i \mathbf{K}(y - x_i) \qquad y \in [1...N]$$

- space subdivision tree structure
  - to find “near” and “far” bodies

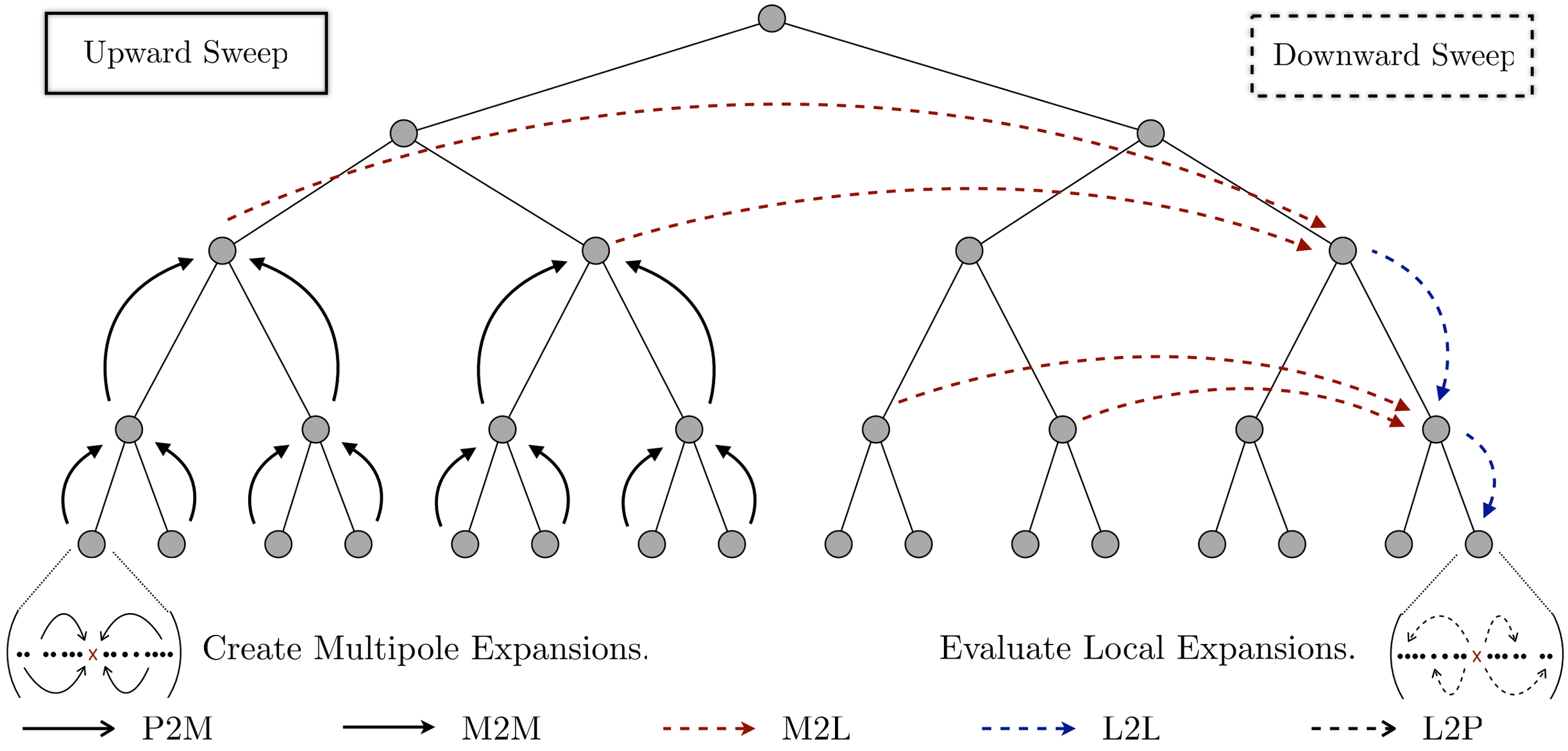




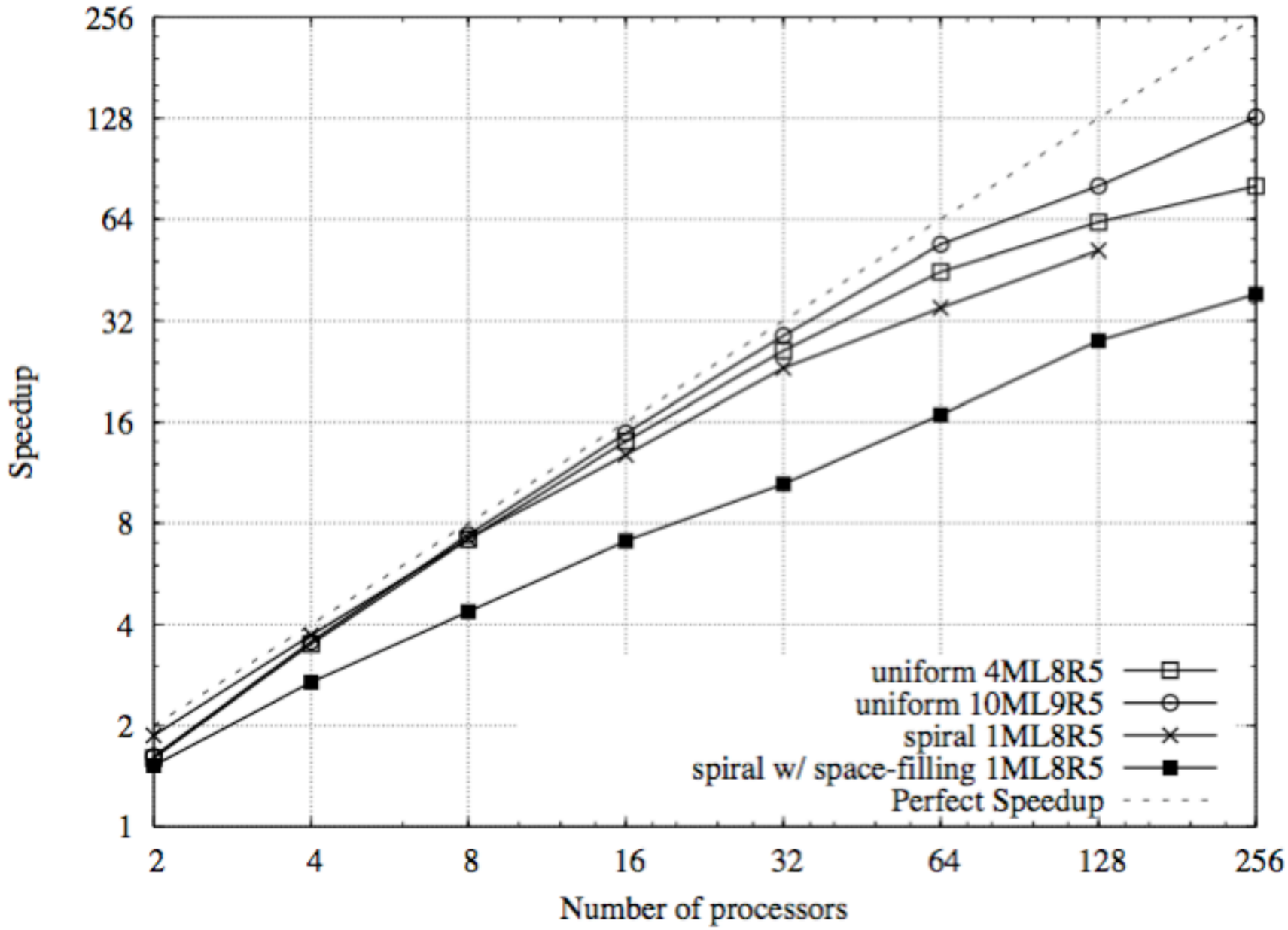




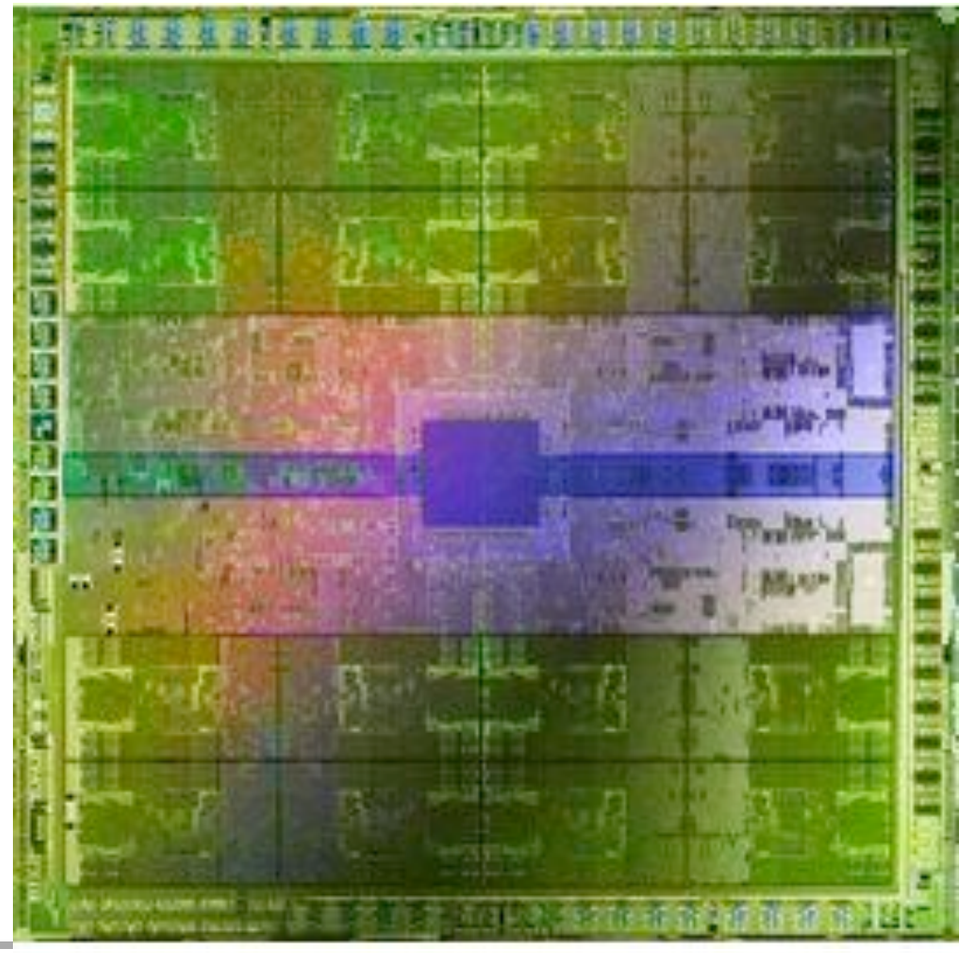
- The whole algorithm in a sketch



- Open-source library: **PetFMM**



Code — `http://barbagroup.bu.edu/Barba_group/PetFMM.html`



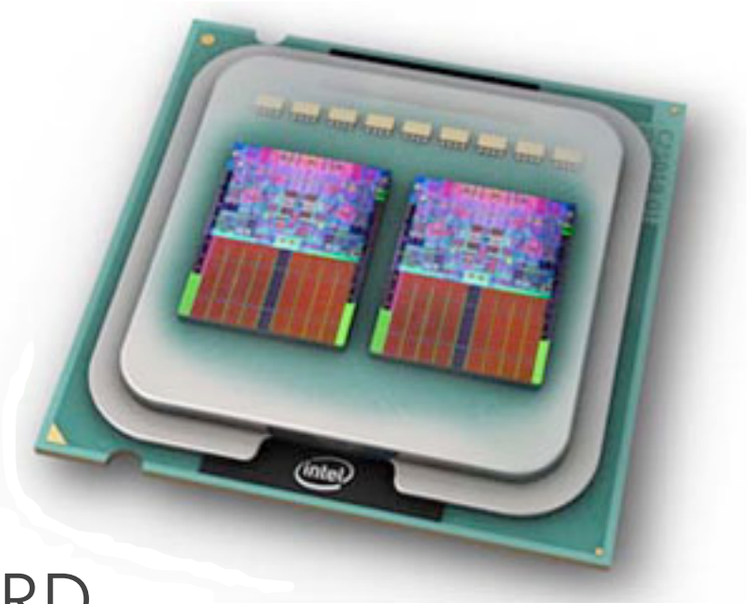
Innovation using new hardware

BU College of Engineering

"Unwelcome advice"

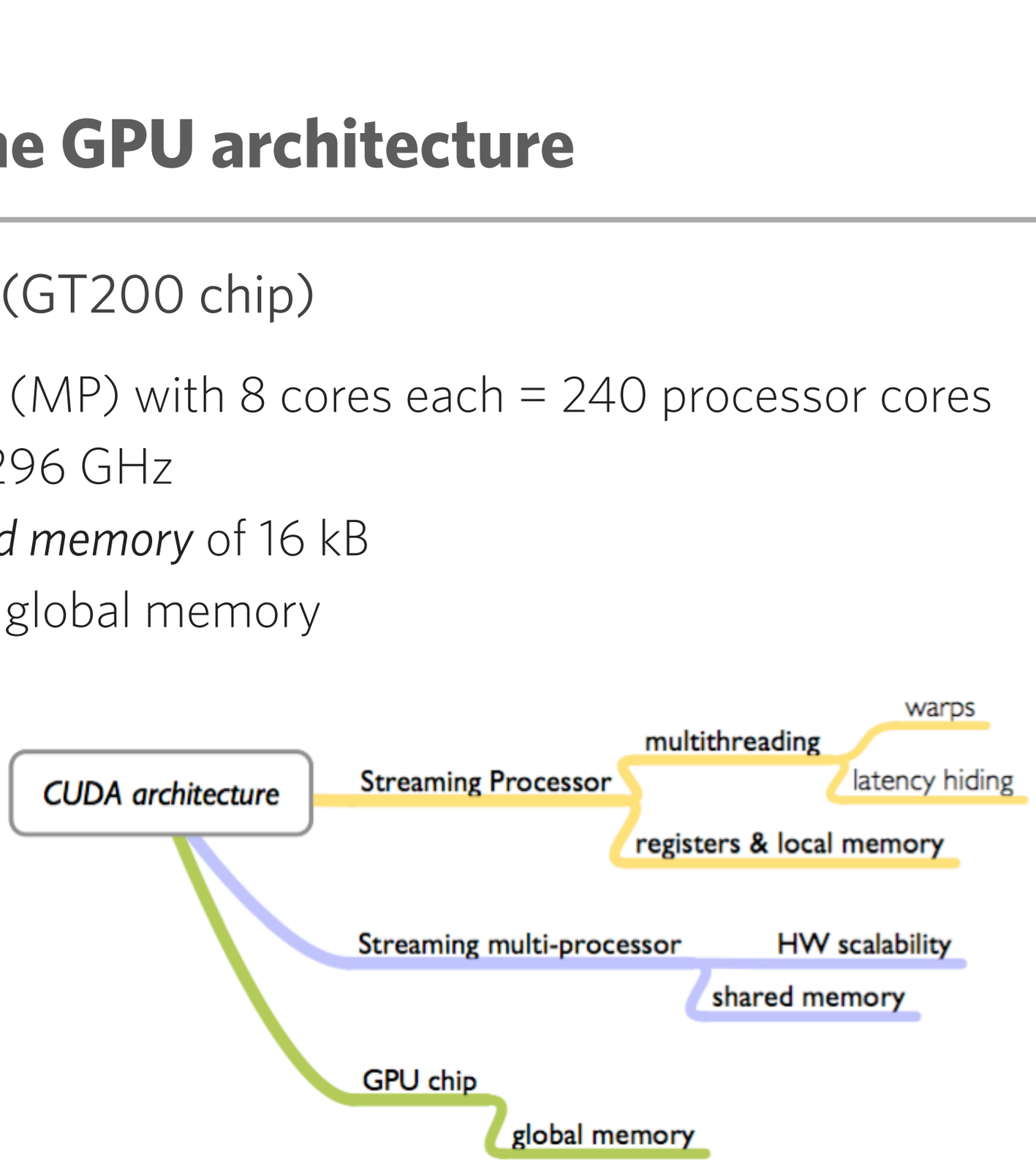
A. Gholum, Intel (June 2008)

- Incremental path:
  - scaling to tap dual- and quad-core performance
    - a flat route (EASY) but leading to nowhere
    - only option for legacy code
- Going *"back to the algorithmic drawing board"* — HARD
  - rethinking core methods
  - 10s, 100s, 1000s of cores
  - basic logic of the application is influenced — design for parallelism



Key features of the GPU architecture

- Nvidia Tesla C1060 (GT200 chip)
  - *30 multi-processors* (MP) with 8 cores each = 240 processor cores
  - cores clocked at 1.296 GHz
  - each MP has *shared memory* of 16 kB
  - device has 4 GB of global memory



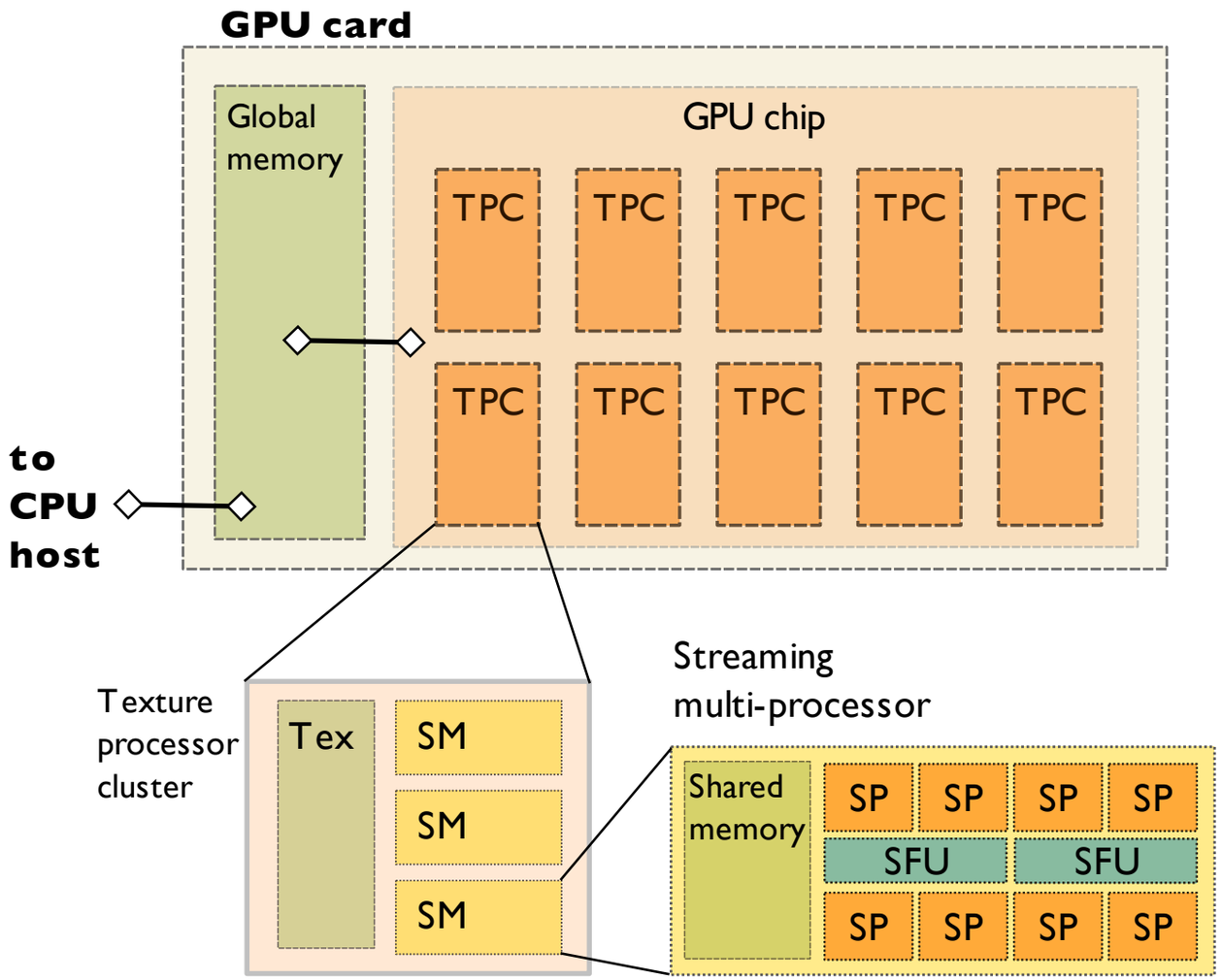
Key features of the GPU architecture

- Nvidia GT200 chip

  1.296 GHz x 10 TCP x 3 SM x 8 SP
  x 2 flop/cycle = 622 Gflop/s

  1.296 GHz x 10 TCP x 3 SM x 2 SFU
  x 4 FPU x 1 flop/cycle = 311 Gflop/s

  Total = 933 Gflop/s



Key features of the CUDA model

Released Feb. 2007

Key features of the CUDA model

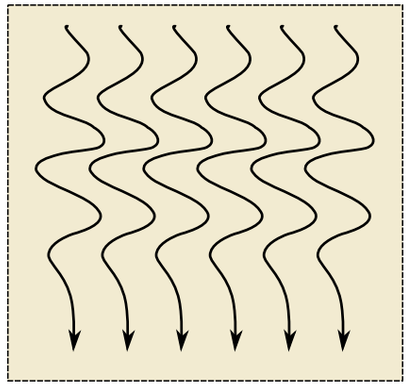
Released Feb. 2007

- "kernels" — work that will be performed by each parallel *thread*

Key features of the CUDA model

Released Feb. 2007

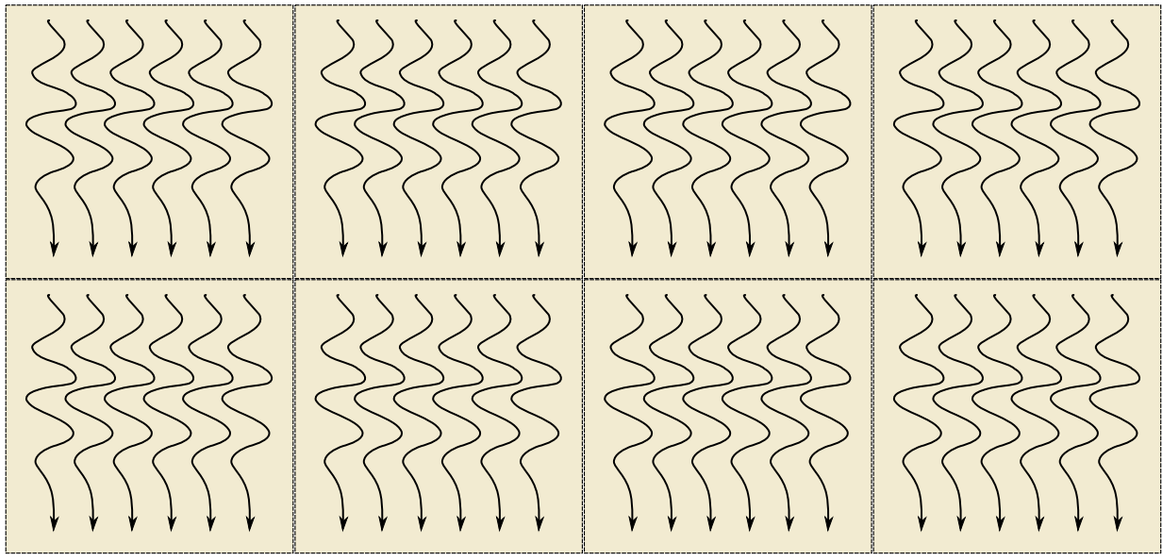
- "kernels" — work that will be performed by each parallel *thread*
  - hierarchy of *thread blocks* and a *grid* of thread blocks



Key features of the CUDA model

Released Feb. 2007

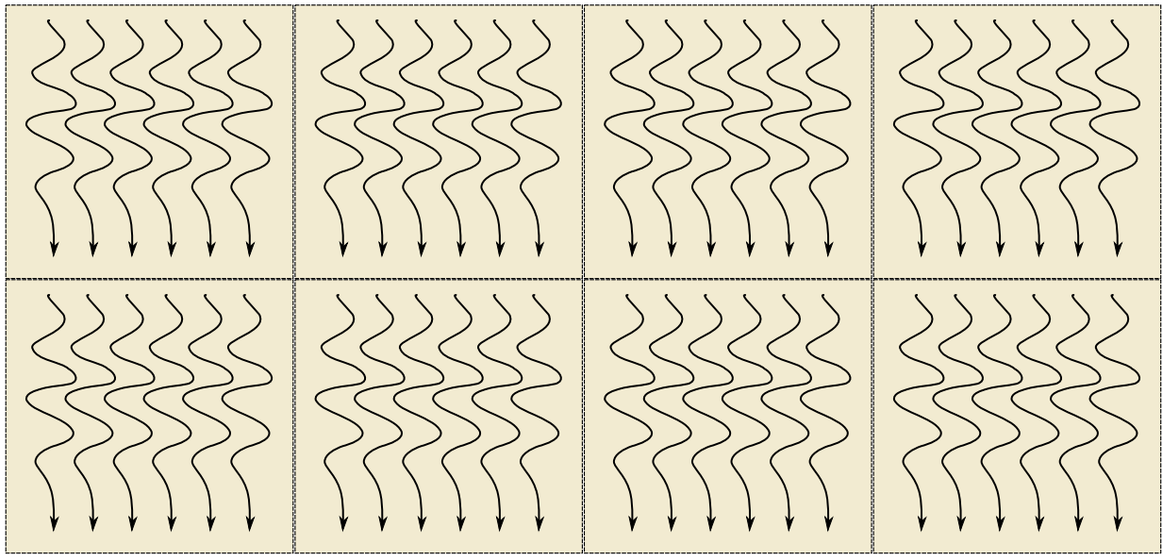
- "kernels" — work that will be performed by each parallel *thread*
  - hierarchy of *thread blocks* and a *grid* of thread blocks



Key features of the CUDA model

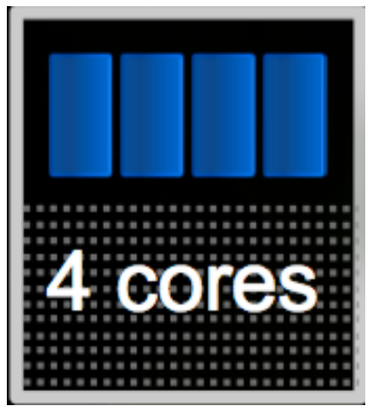
Released Feb. 2007

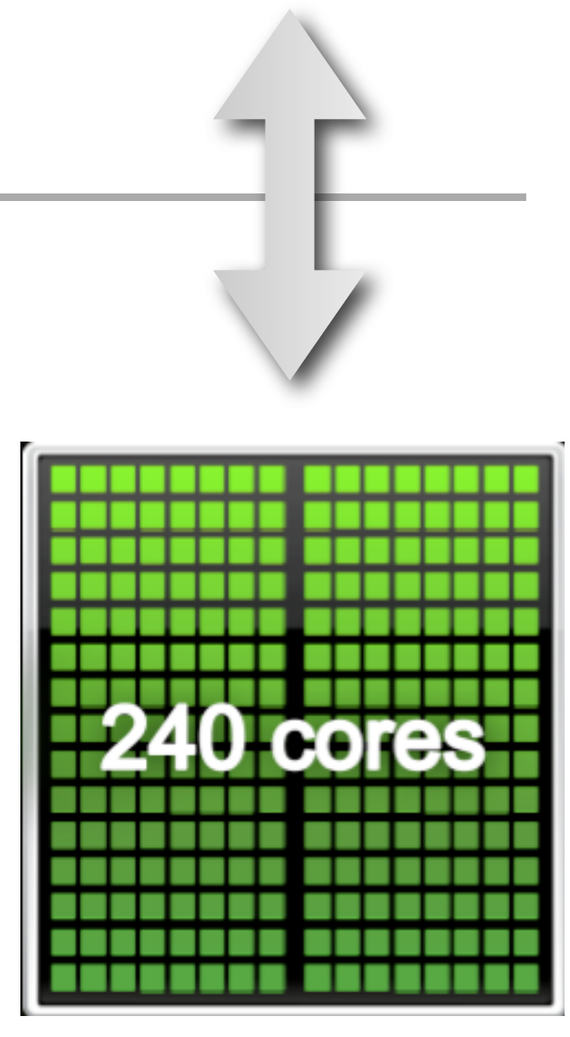
- "kernels" — work that will be performed by each parallel *thread*
  - hierarchy of *thread blocks* and a *grid* of thread blocks



shared memory
(16 kB)

- threads in a thread block reside in the same MP and cooperate via their *shared memory* and synchronization points





Formulating mathematical algorithms for the new architecture

Challenges of the paradigm shift

Challenges of the paradigm shift

- Limiting factor for performance
  - ability of programmers to produce ***code that scales***

Challenges of the paradigm shift

- Limiting factor for performance
  - ability of programmers to produce ***code that scales***
- Rethink algorithms
  - massively-parallel architecture (10k threads)
  - computationally-intensive will be the winner

Challenges of the paradigm shift

- Limiting factor for performance
  - ability of programmers to produce ***code that scales***
- Rethink algorithms
  - massively-parallel architecture (10k threads)
  - computationally-intensive will be the winner
- Implementation difficulties
  - steep learning curve of ***getting down to the metal***
  - memory resources handled by hand
  - data access pattern now crucial

“Resource-conscious” algorithms

- Data-parallel programming
  - Number of threads — thousands !
- Memory
  - shared memory — only 16 kB !
  - global memory — high cost of access: 400-600 cycles
- Branching
  - MPs manage threads in groups of 32, called *warps*
  - In a warp, threads start and end together
  - In branching — warp executes all paths serially !

} affects the entire algorithm

Formulation of the FMM for the GPU

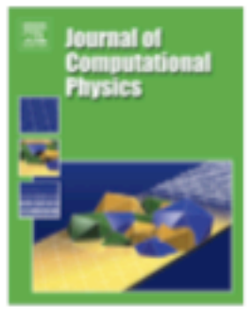
- Recent work:

Journal of Computational Physics 227 (2008) 8290–8313

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp



Fast multipole methods on graphics processors

Nail A. Gumerov *, Ramani Duraiswami

Perceptual Interfaces and Reality Laboratory, Computer Science and UMIACS, University of Maryland, College Park, United States
Fantalgo, LLC, Elkridge, MD, United States

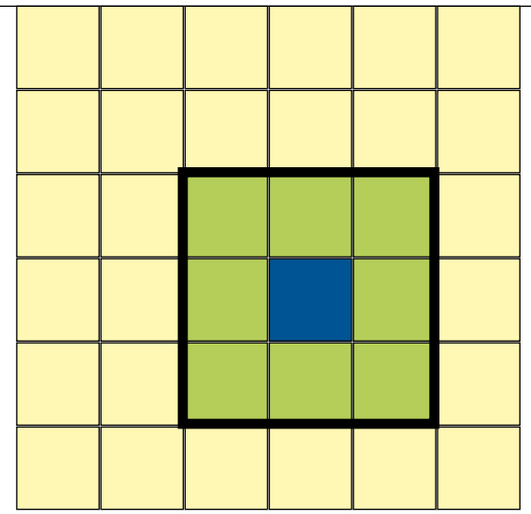
Table 6
Performance for the FMM downward pass

l_{max}	$p = 4$			$p = 8$			$p = 12$		
	CPU (s)	GPU (s)	Ratio	CPU (s)	GPU (s)	Ratio	CPU (s)	GPU (s)	Ratio
3	0.031	0.0146	2.1	0.093	7.71E−02	1.2	0.218	0.233	0.9
4	0.203	0.0614	3.3	0.936	2.65E−01	3.5	2.39	0.718	3.3
5	1.86	0.359	5.2	8.55	1.80	4.8	21.9	4.84	4.5

“going back to the algorithmic drawing board”

Felipe Cruz, PhD student

- M2L = matrix-vector multiplication
  - dense matrix of size equal to $2p^2$
  - large number of mat-vecs — 27×4^L (in 2D)
    - e.g. L=5 $\Rightarrow$ 27,648 mat-vecs

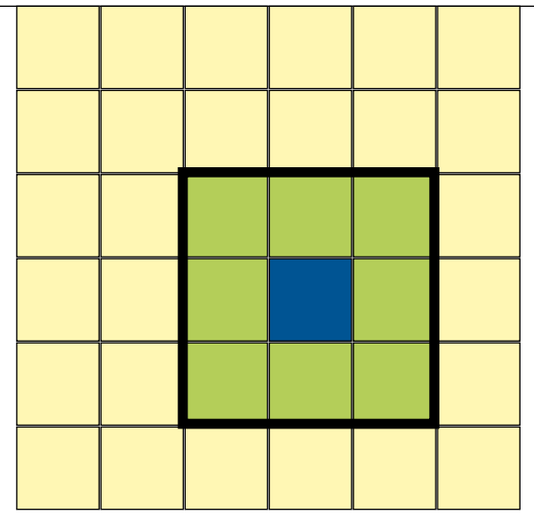


- Version 0 — each mat-vec in a separate thread
  - e.g. p=12, matrix size $2p^2$ =288 $\Rightarrow$ 2304 bytes
    - max. of 6 fit in shared memory — too few threads!

"going back to the algorithmic drawing board"

Felipe Cruz, PhD student

- M2L = matrix-vector multiplication
  - dense matrix of size equal to $2p^2$
  - large number of mat-vecs — 27×4^L (in 2D)
    - e.g. L=5 $\Rightarrow$ 27,648 mat-vecs

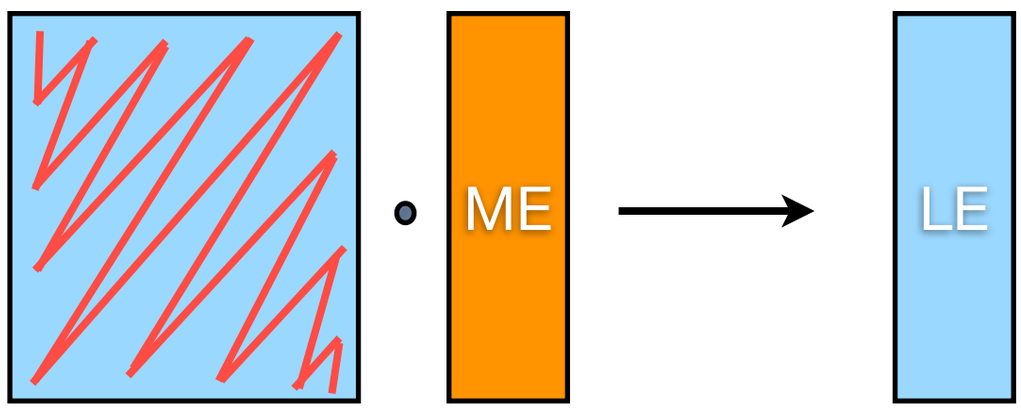


- Version 0 — each mat-vec in a separate thread
  - e.g. p=12, matrix size $2p^2$ =288 $\Rightarrow$ 2304 bytes
    - max. of 6 fit in shared memory — too few threads!

} compute on-the-fly

“going back to the algorithmic drawing board”

- structure of matrix $$a_{nk} = (-1)^n \binom{n+k}{k} t^{-k-n-1}$$
- optimization:
  - matrix-free mat-vec
  - traverse matrix by diagonals
  - reuse terms



“going back to the algorithmic drawing board”

- Version 1 : each thread block transforms one ME for all the interaction list.
  - max 8 concurrent thread blocks per MP ⇒ 27x8= 216 concurrent threads
  - current maximum of threads is 512 — still under-utilized

Result:

20 Gflops peak

2.5×10^6 t/s

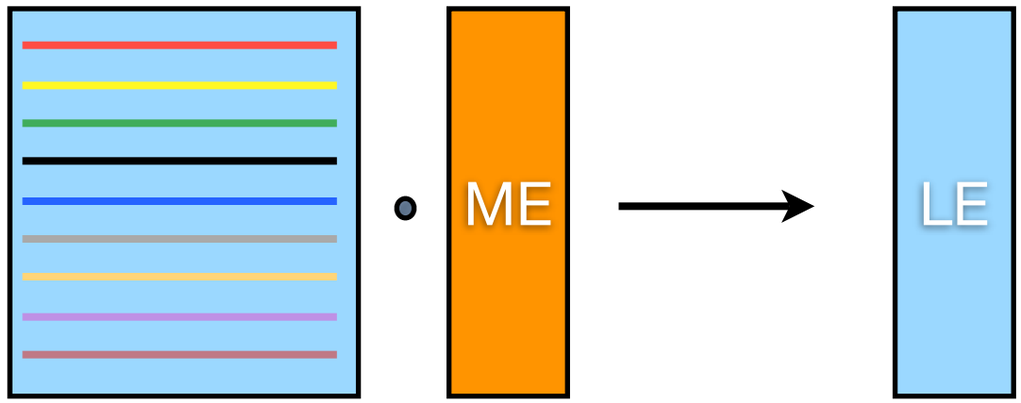
We are not reporting speed-up anymore, because it does not mean very much.
“A CPU run” produced 1.42×10^5 t/s

"going back to the algorithmic drawing board"

- we're still not happy —want more parallelism
- too many memory transactions
  - result moved to global memory — one LE, size 2p
  - 27 results ⇒ 27 x 2p = 648 floats for p=12
  - only 32 "workers" to move results ⇒ 20 memory transactions

"going back to the algorithmic drawing board"

- Version 2 : one thread per *element* of the LE
  - each thread does 1 row-vec multiply
  - no thread synchronization required
  - BUT — cannot use the efficient "diagonal traversal"
    (about 25% more work)



"going back to the algorithmic drawing board"

- Version 2 : avoiding ***branching***

$$a_{nk} = (-1)^n \binom{n+k}{k} t^{-k-n-1}$$

  - each thread pre-computes t^{n+1} and additional t factors added as needed
  - loop over n in each thread: naturally would stop at different value
  - instead (counter-intuitive!), all threads loop until $n = p + 1$ but ***store*** only their relevant power

- no optimizing/tuning yet ...

Result:

94 Gflops peak

6×10^6 t/s

key features of new algorithm

1. increased *number of threads* per block (we can have any number)
2. all threads always perform the *same computations* (on different data)
3. *loop-unrolling* (if done manually, performance gain even greater!)
4. reduced accesses to global memory
5. when accessing memory, ensure it is *coalesced*

- Final step: overlap memory accesses

Result:

480 Gflops peak

19×10^6 t/s

Context of this project

- Fast summation algorithms
  - we have a distributed parallel library of the fast multipole method, PetFMM (*)
  - developing GPU implementations
    - currently running at about 500 gigaflops on one card
    - speedups w.r.t. fastest CPU available.
- Applications:
  - meshfree fluid simulation, using N-body solvers

(*) In collaboration with Matthew Knepley (UChicago)

Frontiers of CFD

- Need to straddle many scales
- Algorithms to detect and adapt to solution
  - meshfree methods naturally can adapt
- Hardware-aware software
  - meshfree methods well-suited for GPU!
- Tackle problems with complex/moving geometry
- Algorithm/software that allows real-time simulation